User's Manual

3CX Voice Application Designer

Version 14

Copyright 2006-2015, 3CX Ltd.

http://www.3cx.com

E-mail: info@3cx.com

Information in this document is subject to change without notice. Companies' names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of 3CX Ltd.

3CX Phone System for Windows is copyright of 3CX Ltd.

3CX is a registered trademark, and 3CX Phone System for Windows and the 3CX logo are trademarks of 3CX Ltd. in Europe, the United States and other countries.

Version 14 – Last updated 10/11/2015

Table of Contents

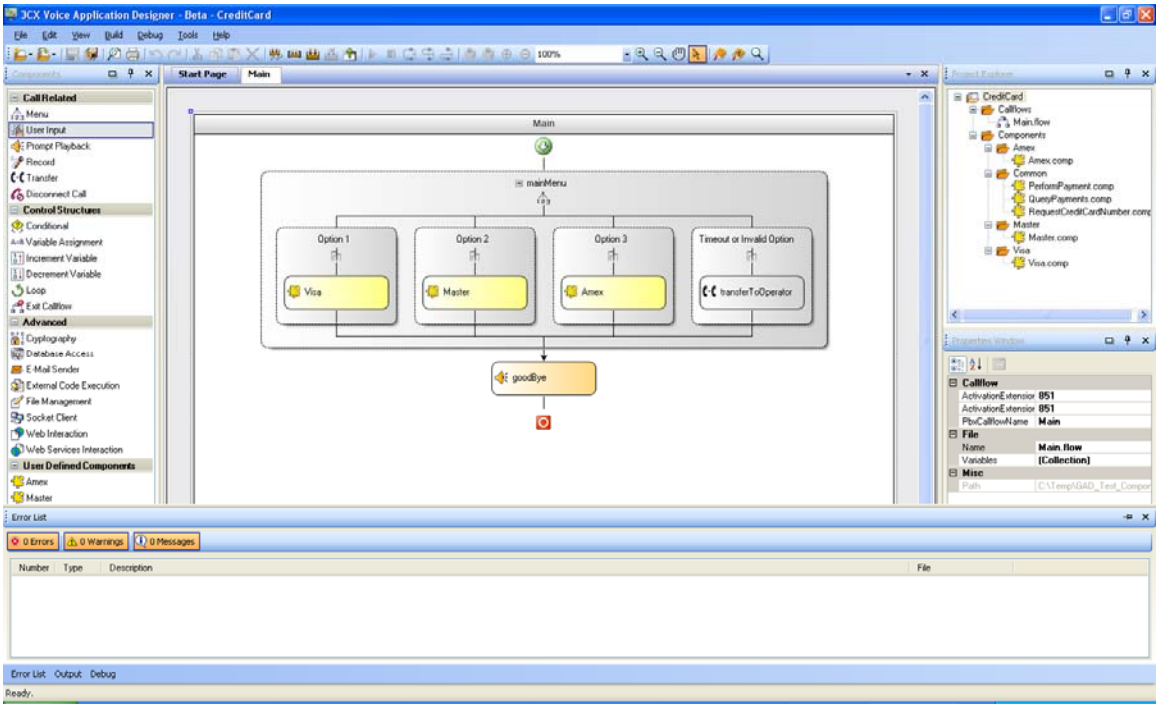
Introduction 1
What is 3CX Voice Application Designer? 1
How does it work? 1
Terminology 2
Document conventions 2
Installing 3CX Voice Application Designer 3
System Requirements 3
Run set-up 3
Activating 3CX Voice Application Designer 3
Configuring Phone System location 4
Getting started – Creating a simple callflow 6
Introduction 6
Creating the project 6
Creating a callflow 7
Adding components to the callflow 8
Building the project 13
Deploying the project to 3CX Phone System 13
Direct mode 13
Creating a deployment package 14
Executing your voice application 14
3CX Voice Application Designer Workspace 16
Introduction 16
The Designer 16
Navigating within the designer 17
Types of call flows 17
The Docking panels 20
Components Toolbox 21
Table 3: Component description 21
Project Explorer Window 22
Properties Window 22
Error List Window 23
Output Window 24
Debug Window 24

Components 25
Introduction 25
Prompt Types 25
The Expression Editor 27
Call Related Components 29
Menu 29
User Input 30
Prompt Playback 32
Record 33
Transfer 34
Disconnect Call 35
Control Structures Components 35
Conditional 35
Variable Assignment 35
Increment Variable 36
Decrement Variable 37
Loop 37
Exit Application 37
Advanced Components 38
Cryptography 38
Database Access 39
E-Mail Sender 41
External Code Execution 42
File Management 44
Socket Client 45
Web Interaction 46
Web Services Interaction 47
User Defined Components 49
Default Component options 49
Menu 50
Prompt Playback 50
Record: 50
Transfer 50
User Input 51
Cryptography 51

Database Access: 52
E-Mail Sender: 52
External Code Execution: 53
File Management: 53
Socket Client: 53
Web Interaction: 53
Web Services Interaction 53
Debugging 54
Introduction 54
Building in Debug mode 54
Callflow debugging 54
Sample Application 59
Introduction 59
Creating and configuring the Callflow 59
Creation of the Project and the “Main” Callflow 59
Configuring the Main Callflow 62
Configuring User Components variables 64
Designing User Components 65
Designing QueryPayments Component 65
Designing PerformPayment 69
Designing RequestCreditCardNumber component 72
Error Handler for the Visa Component 74

Introduction

What is 3CX Voice Application Designer?



Screenshot 1 - The voice application designer

The 3CX Voice Application Designer is a visual design tool that allows you to easily create voice applications. Voice applications are programs that interact with the user via the phone and then perform certain logic. For example, you could create a voice application that asks for a caller to input a customer number, which is then verified against a database. Based on customer number a call can be routed to a certain queue.

Because the 3CX Voice Application Designer is entirely visual, little telephony or programming knowledge is required.

How does it work?

Using the 3CX Voice Application Designer, you visually create your application. Each application consists of a number of building blocks, called components. For example ‘Menu’, ‘Prompt playback’, ‘User input’ are a few of the most common components. Each component has particular configuration options. When you have put together your voice application, it is as a project. This project is then uploaded to 3CX Phone System. Using a digital receptionist or a DID number you can then ‘link’ to your voice application. In that way, callers can be directed to your application either directly or via a menu. From within

voice applications you can link back to call queues, extensions, digital receptionists and so on.

Terminology

The 3CX Voice Application Designer or VAD, uses a number of terms:

- **Project**: - your actual application, composed of callflows and components.
- **Callflow** – a particular interaction with a caller, i.e. a group of actions formed by a group of components. For example, asking for a user's customer number.
- **Components**: the basic building blocks of a call flow, such as play a menu, get user input and so on. The main difference between callflows and components is that callflows can be the starting point of a call, while components cannot. Components are always a part of a callflow or a more complex component.
- **Custom Component** – You can create custom components with custom functionality.

Document conventions

The following conventions have been established to improve reading comprehension:

Bold	It is used to highlight key words or concepts.
Bold	It is used to identify the names of the product.
`Courier Type`	It is used to establish application code, file names, classes and / or DB tables.
	It is used to remark a paragraph that contains additional information that should be considered.
n/a	Not applicable.

Installing 3CX Voice Application Designer

System Requirements

The 3CX Voice Application Designer requires the following:

- Windows 7, Windows 8, Windows 8.1, Windows 2008 R2 server, Windows 2012 Server or Windows 2012 R2 Server
- .NET Framework version 4.0 or higher
- 1 Gigabyte Memory or higher, Pentium 4 processor or up
- 3CX Phone System standalone version 12.5 or up

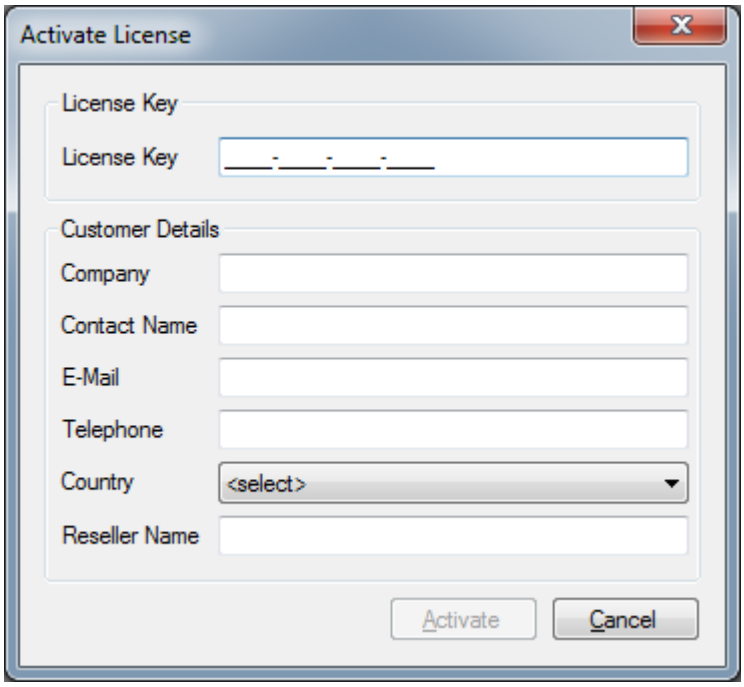
Run set-up

The following steps describe how to install the 3CX Voice Application Designer in your computer:

- Run set-up by double-clicking on the set-up file. Click ‘Next’ to start installation.
- Select the installation location and choose to install the application for the current user or every user in the computer. 3CX Voice Application Designer will need a minimum of approximately 15 MB of free hard disk space. You will need to reserve additional space to store projects. Click ‘Next’ to continue.
- Click ‘Next’ to confirm the installation. Setup will now copy all files to the destination folder. To complete the install click ‘Close’.

Activating 3CX Voice Application Designer

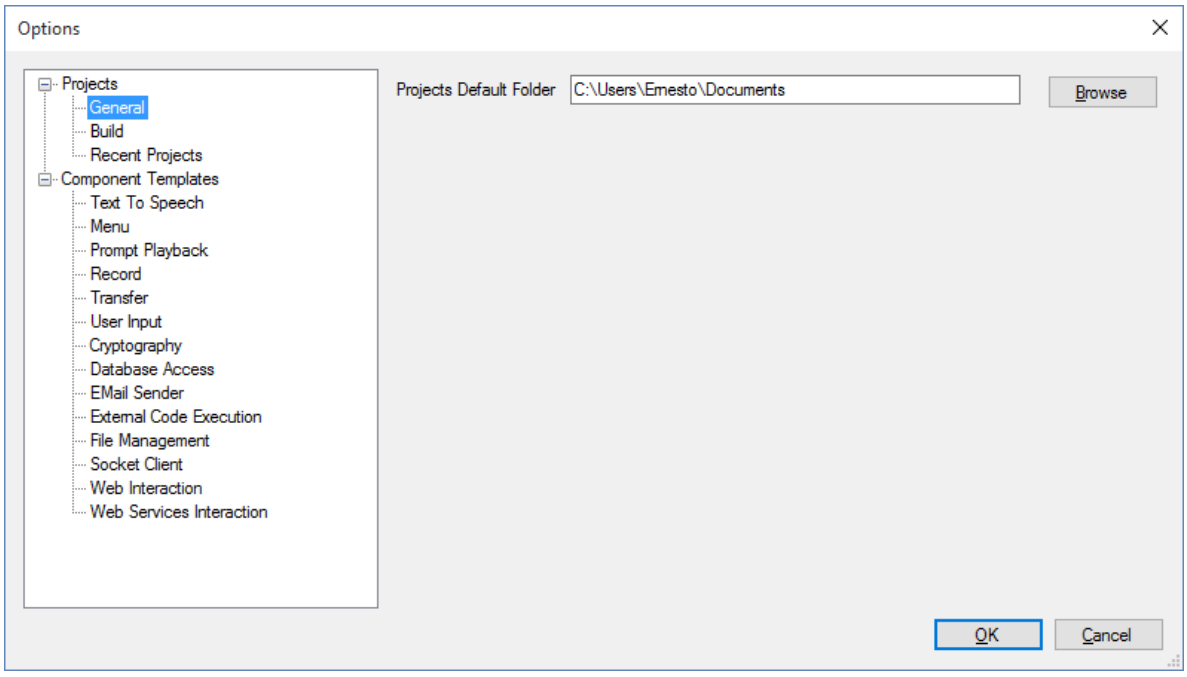
If you have purchased 3CX Voice Application Designer, then you can activate your license by going to the Tools → Activate License.



Enter your license key, Company, Contact Name, E-mail, Telephone, Country and Reseller Name, and click on “Activate” to activate your license. This information will be sent to our license key server and your license key and installation will be activated.

After activating your license, in order to use the Text to Speech functionality you will have to restart the 3CX Voice Application Designer.

Configuring Phone System location



Screenshot 2 - Options – Projects – General

After you have installed 3CX VAD, you need to specify the location where you will save your projects:

1. Go to Tools → Options from the main menu. Here you can configure general application parameters, not related to a specific Project.

2. Configure

- **Project Default Folder**: this is the default folder used to save the projects when a user opens or creates a new Project.

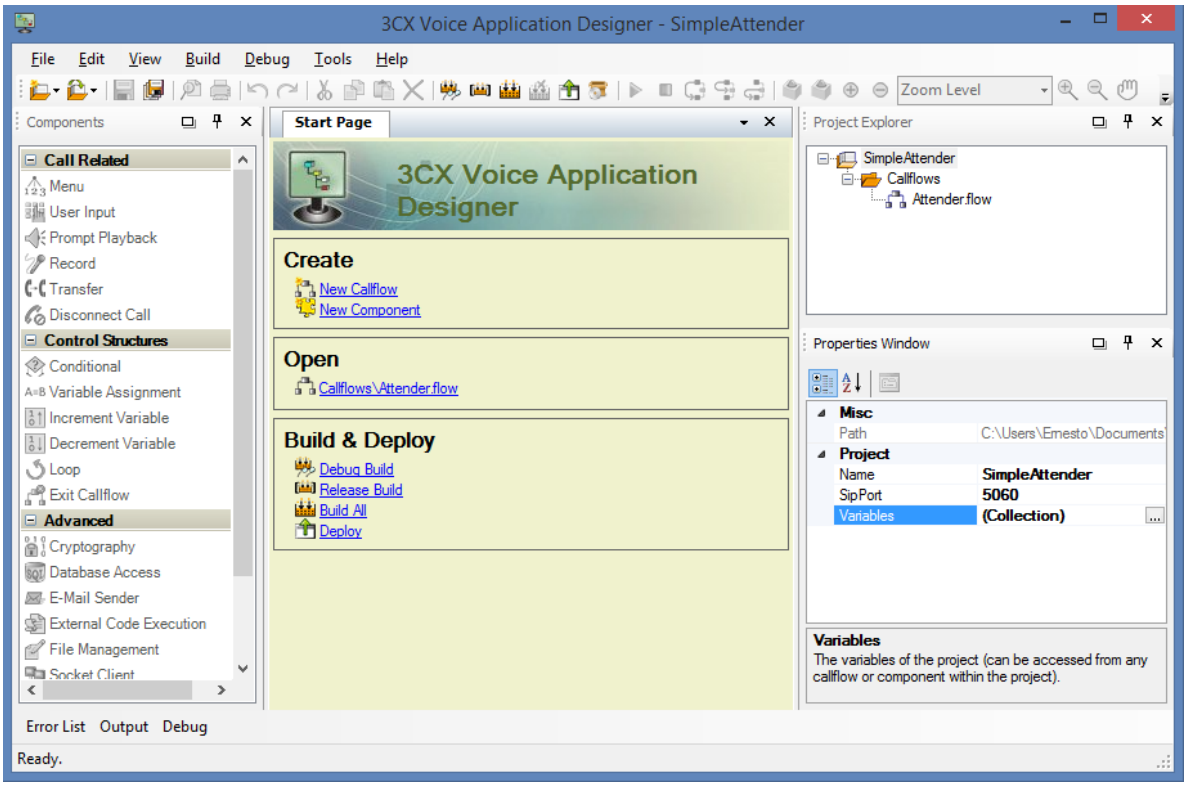
Getting started – Creating a simple callflow

Introduction

In this chapter we'll explain how to create a simple callflow and deploy it to 3CX Phone System. The idea of this chapter is to give an overview of how to work with the VAD. After you are able to build a simple application you can then familiarize yourself with the more advanced components and build more complex applications.

In this chapter we will build a callflow which will offer a simple menu (For Sales press 1, for Help Desk press 2, to make an outbound call press 3, or hold on and an operator will assist you). The concepts used in this section are explained in detail in the following chapters.

Creating the project



Screenshot 3 - Creating a project

The first step is creating the project. To do this:

1. Click File → New → Project

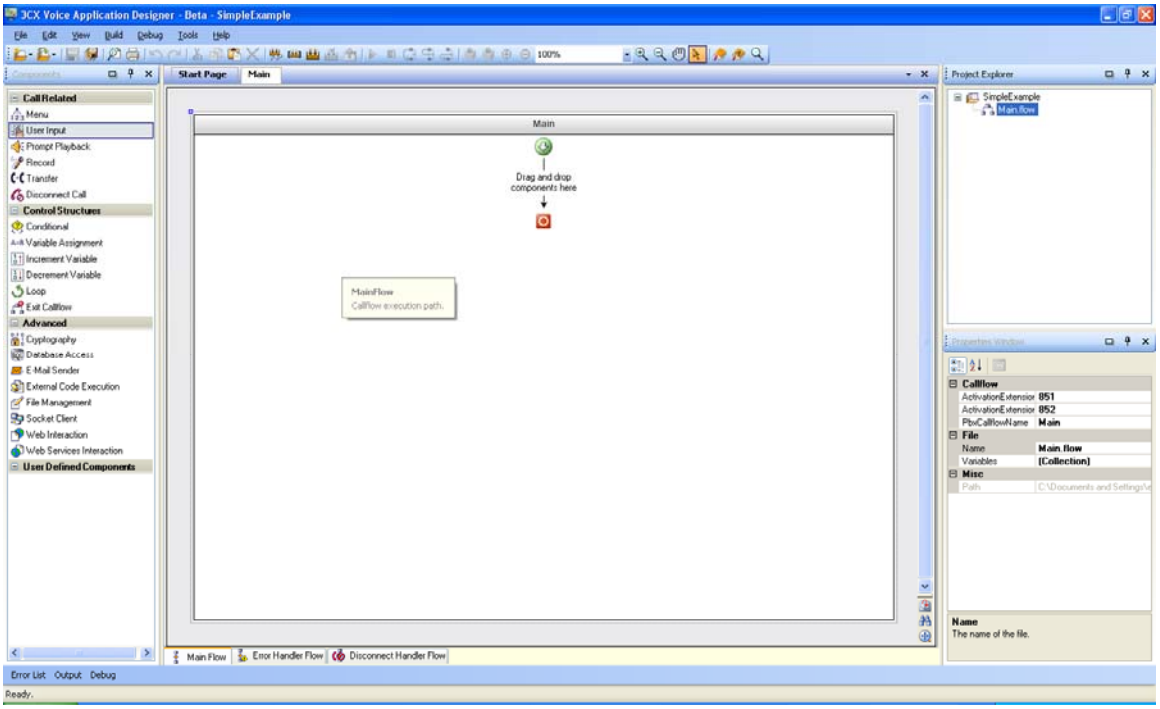
2. Enter a project name and location and click Save.

3. The project will be created with default properties. If you need to change them, select the project in the Project Explorer. Then the Properties window will show its properties and let you change them.

The application will create a folder to store audio files for the project, another folder to store library files for the project, and another folder to create deployment packages. It will also create a folder to generate the output of the building process in debug and release modes.

The project will be created according to the default properties for a project. You can change this from Tools → Options → Projects → General.

Creating a callflow



Screenshot 4 - Creating a call flow

When the project is created, an empty “Main” callflow is automatically created too. A new tab will open on which you can drag the components that will form your call flow.

Each callflow will be created with three basic properties, which can be edited from the properties Window:

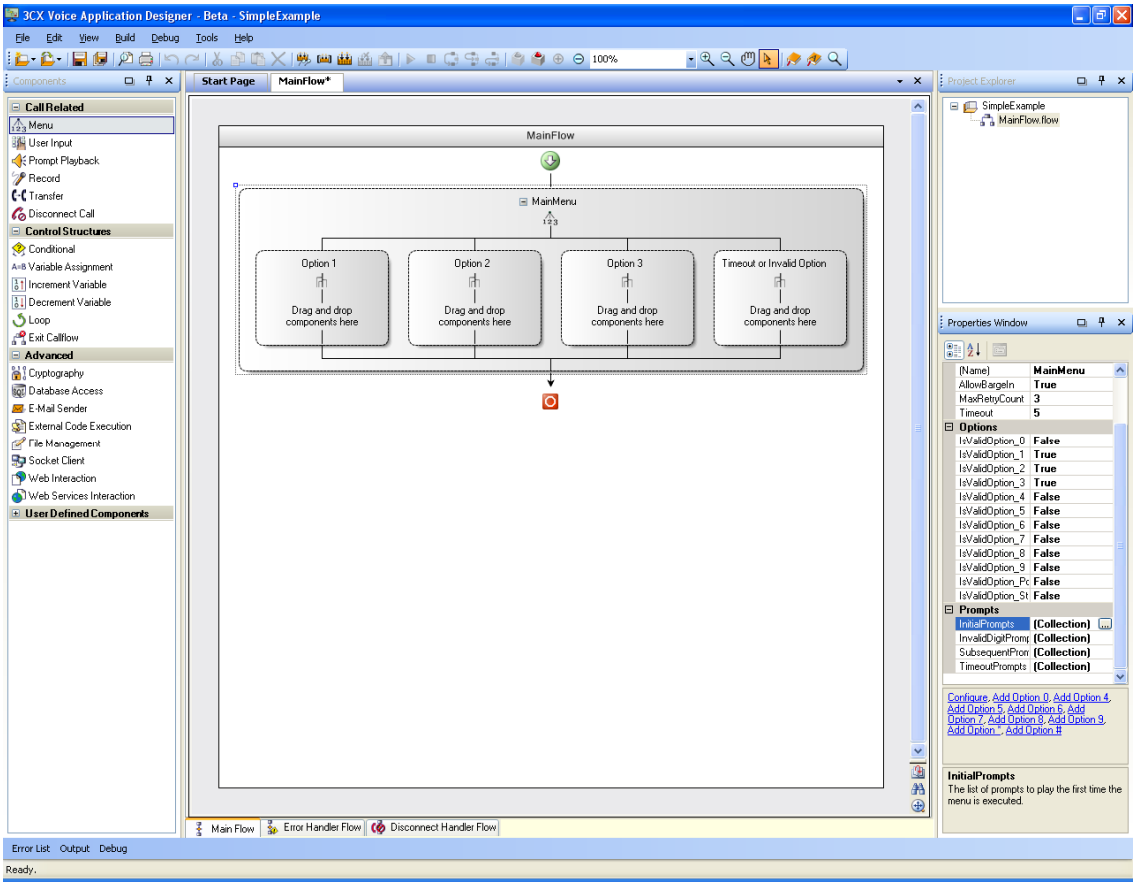
ActivationExtension: The extension number on which the application will register this callflow in 3CX Phone System when the project is deployed in release mode. You will be able to call this extension from any extension in 3CX to execute this callflow. You can also configure an extension range here, in case you need to deploy the callflow to many extensions at a time. For example, if you set this property to 851, then the callflow will be registered on that extension, or you can set it to 851-859 to register the callflow in that 10 extension range.

ActivationExtensionDebug: The extension number where the application will register this callflow in 3CX Phone System when the project is deployed in debug mode. You will be able to call this extension from any extension in 3CX to execute this callflow. You can also configure an extension range here, in case you need to deploy the callflow to many extensions at a time. For example, if you set this property to 851, then the callflow will be registered on that extension, or you can set it to 851-859 to register the callflow in that 10 extension range.

PbxCallflowName: it is the name that will be used to register this callflow in 3CX Phone System. You will see this callflow in the 3CX management interface, listed as a Digital Receptionist with this name.

To show the call flow, double click on the callflow in the Project Explorer.

Adding components to the callflow



Screenshot 5 - Adding components

Once you have created the callflow, you can drag and drop components from the toolbox to the designer. In this example, we'll add a Menu component:

1. From the toolbox on the left hand side, select the component 'Menu'

2. Now drag it onto one of the options of the call flow. Drag it onto 'Drag and drop component here'. You will see 'Menu component' written when you have dragged it on to the correct location.

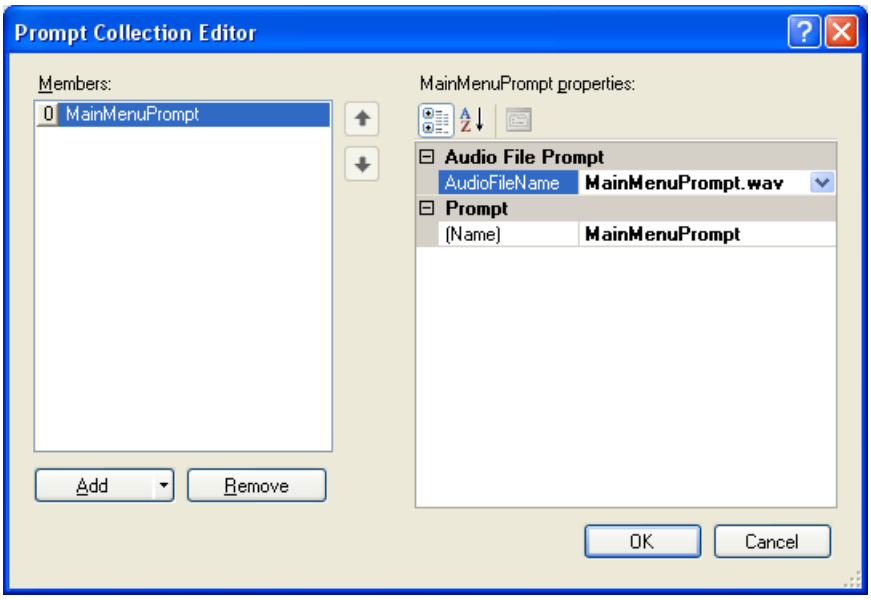
3. Now give the component an appropriate name. In this case we'll name it "MainMenu".

4. By default each call flow starts with 2 options. We're going to add a third option. Right- click with your mouse anywhere in the designer window and select "Add Option 3".

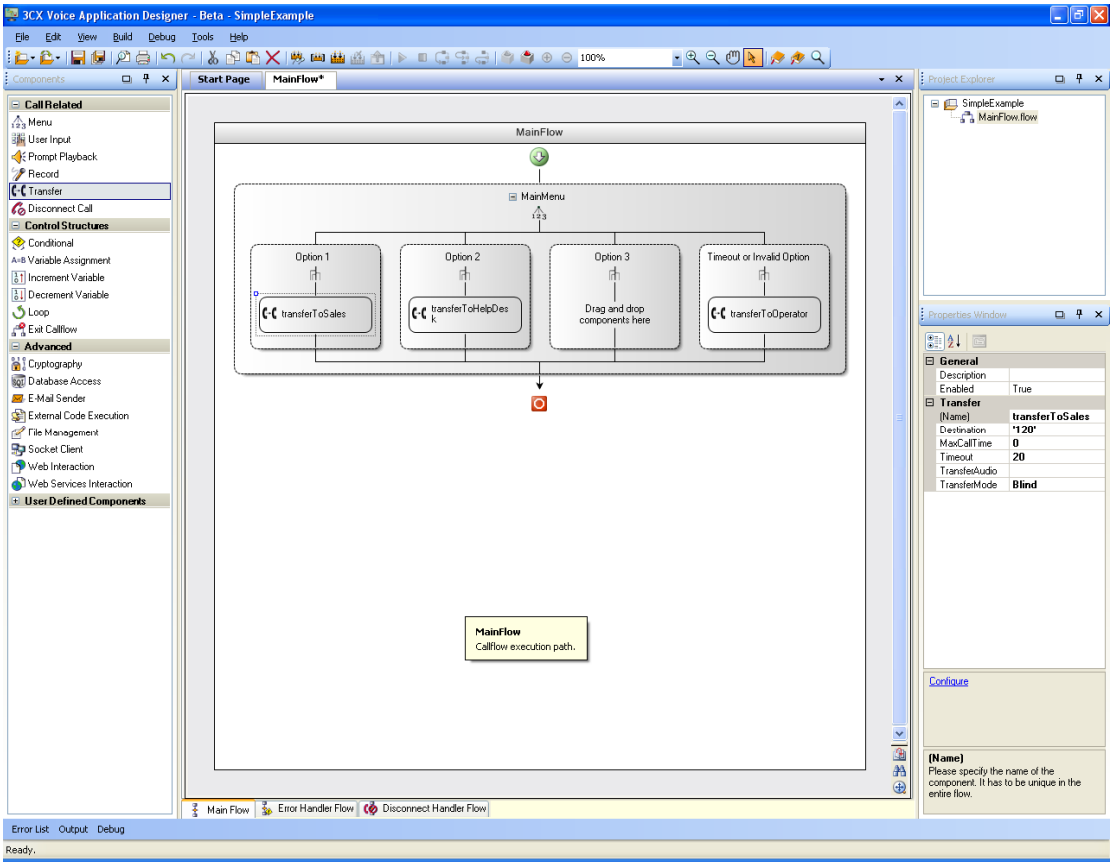
Note: When the Menu component is added to the designer, it will have its properties set to the template you have in Tools → Options → Component Templates → Menu. You may change the Menu component properties from the Properties windows or its configuration form. The configuration form can be opened by double clicking the component or in the context menu displayed when you right click on it.

5. To change the menu prompt, edit the "InitialPrompts" property from the Properties window.

The Prompt Collection Editor will be shown. There you may add an "AudioFilePrompt", and select an audio file from the drop down list. This list shows the names of the files you have in the audio folder of the project. You need to record the prompt and copy the file to that folder in order to get it listed here. Audio files can be wav or mp3 format.



6. You may want to add prompt messages to play when the user enters an invalid option or does not input anything. You can do that by changing the "InvalidDigitsPrompts" and "TimeoutPrompts" properties for the Menu component.

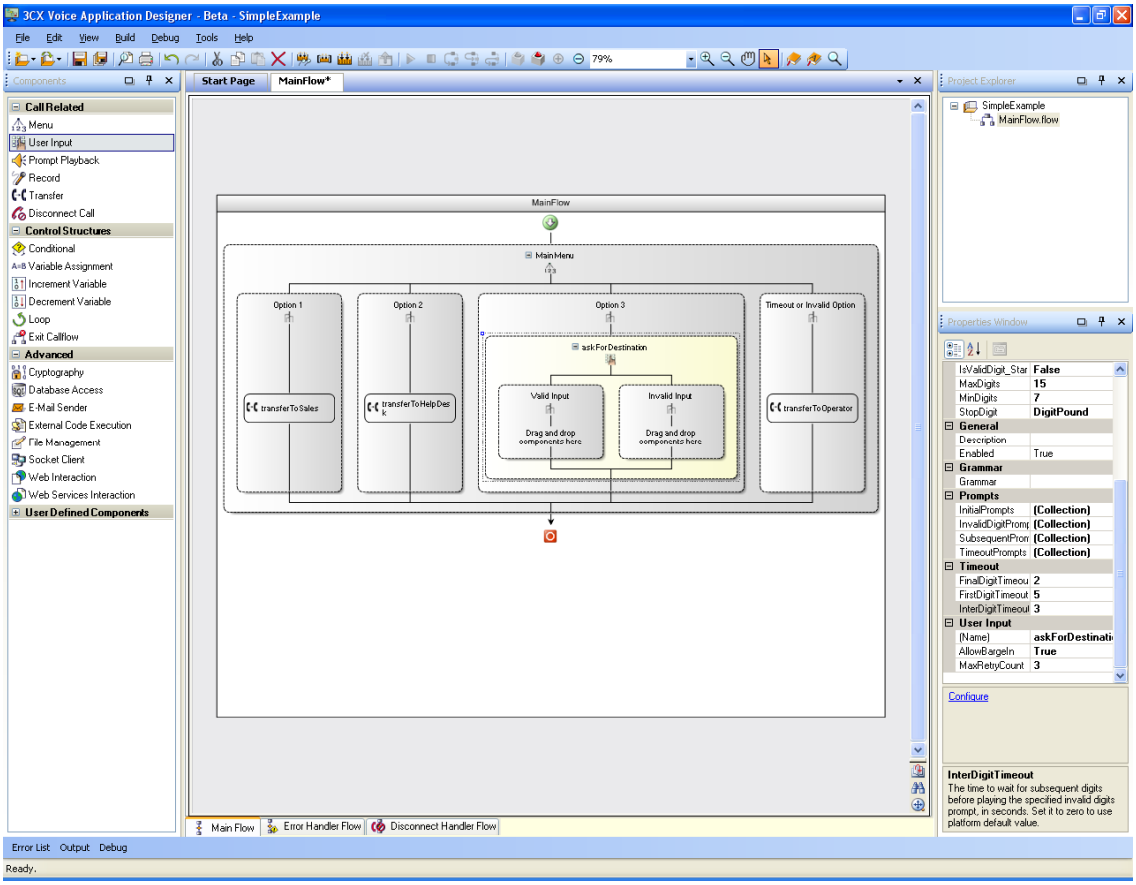


Screenshot 6 - Configuring menu option actions

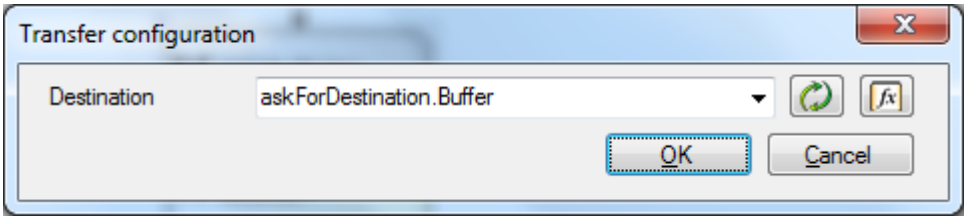
7. Now we configured the simple menu and we need to add the components that will be executed when any of the options is activated. Let's assume we want to transfer to the Sales department if the user presses 1. To do this, drag and drop a Transfer component from the Toolbox to the designer, under the "Option 1" branch.

8. Change the name of the transfer component

9. Double-click on the component and select the sales extension as the destination. Repeat this procedure for option 2 (transfer to Help Desk) and timeout or invalid option (transfer to the operator).



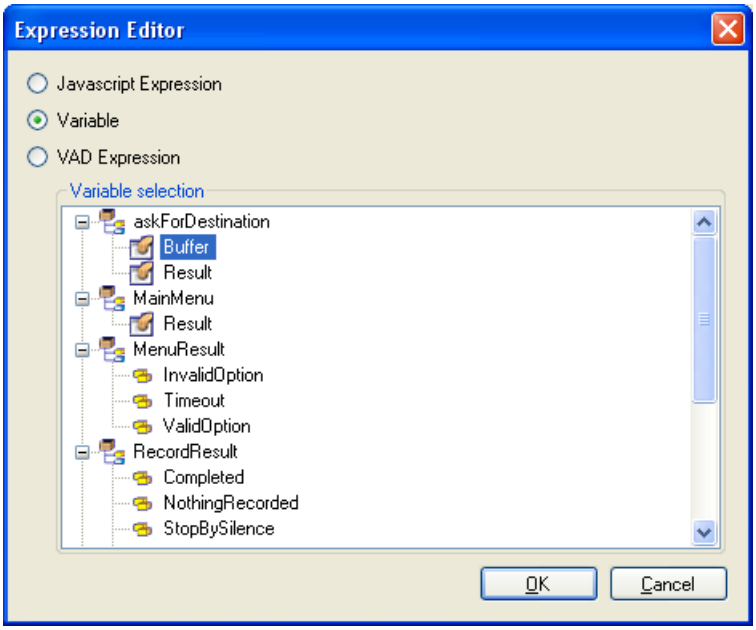
10. For option 3, we are going to make things a little more interesting. We will ask for a destination number and then transfer to that number. To ask for the destination number we will use a User Input component. Drag and drop it under the "Option 3" branch. Change the component name and configure the message prompt to use. Specify that valid digits are numbers from 0 to 9, and other digits are invalid (star and pound). Also set MinDigits and MaxDigits to a reasonable value, for example, between 3 and 5. When the user enters a valid input, we need to transfer the call to the specified destination. Otherwise, we'll transfer the call to the operator.



Screenshot 7 - Transfer configuration

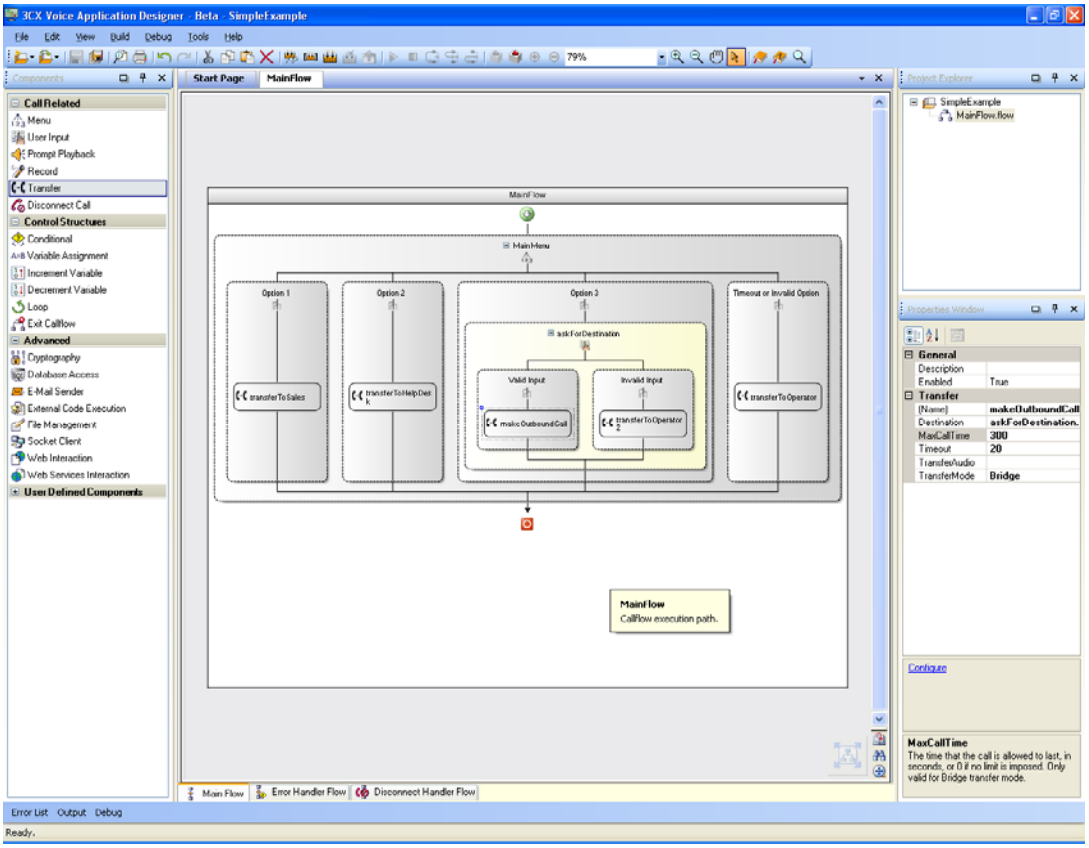
11. Drag and drop a transfer component onto the valid option area. Now double click on the transfer component. The transfer configuration dialog will appear.

12. Now click on the Expression editor button, which is the 'fx' button next to the destination field. The Expression editor will appear.



Screenshot 8 - The expression editor

13. To make an outbound call to the selected destination, select the “Buffer” property of the “askForDestination” component.



Screenshot 9 - The voice application

Building the project

The callflow is ready. Now we need to build it to a VXML script that can be used by 3CX Phone System. The project can be built in two modes:

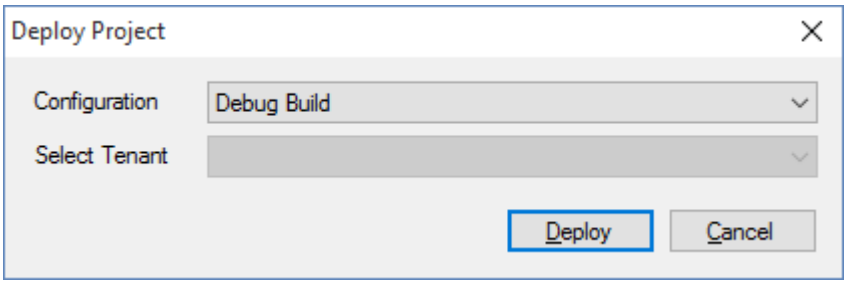
- **Debug mode**: when the project is built in debug mode, the generated output contains code that writes debug information files when a call is being processed. These files can be retrieved later by this application to show the callflow behaviour graphically.
- **Release mode**: when the project is built in release mode, the generated output does not generate debug information files. The execution of a generated application in release mode is more performing, and is suggested for production environments.

In order to build the project, execute the menu command Build → Build All. This option builds the project in both configurations (debug and release). The output window will show compilation progress, and the Error List window will show any error, warning or important message that you might need to see.

Deploying the project to 3CX Phone System

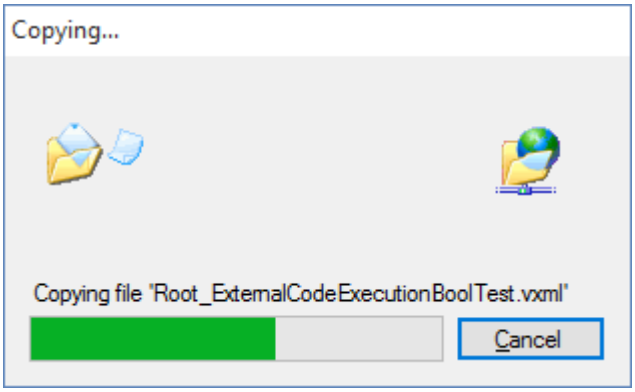
Direct mode

Once you successfully built the project in some configuration (debug or release), you need to deploy it. The project is deployed to the local 3CX installation.



Screenshot 10 – The Deployment Dialog

To deploy, execute the menu command Build → Deploy. The deployment dialog will appear, allowing you to select the desired configuration (release/debug build). If you're deploying to a 3CX Virtual Server (multi-tenant), the list of available tenants will be automatically enabled. Press the Deploy button to start.



Screenshot 11 - Deploying your voice application

After the deployment has been completed, you will see the callflows registered in 3CX Phone System as Digital Receptionists. You can use the 3CX management interface to choose when to activate each callflow, for example when an inbound call arrives from the PSTN.

Creating a deployment package

Sometimes deploying the project to 3CX Phone System in direct mode is not possible. For example, if you're developing the voice application for a customer. In these cases, you may need to use the Create Deployment Package functionality.

With the Create Deployment Package functionality, you can create a package with all the necessary files to deploy your project locally to another server. To do this:

1. Execute the menu command Build → Create Deployment Package
2. Select the desired configuration between Debug and Release
3. Select the output folder, i.e. the folder where the package will be created
4. Press the OK button to create the package
5. Copy the entire output folder to the destination 3CX server
6. In the destination 3CX server, execute the DeployVADProject.exe program to deploy locally

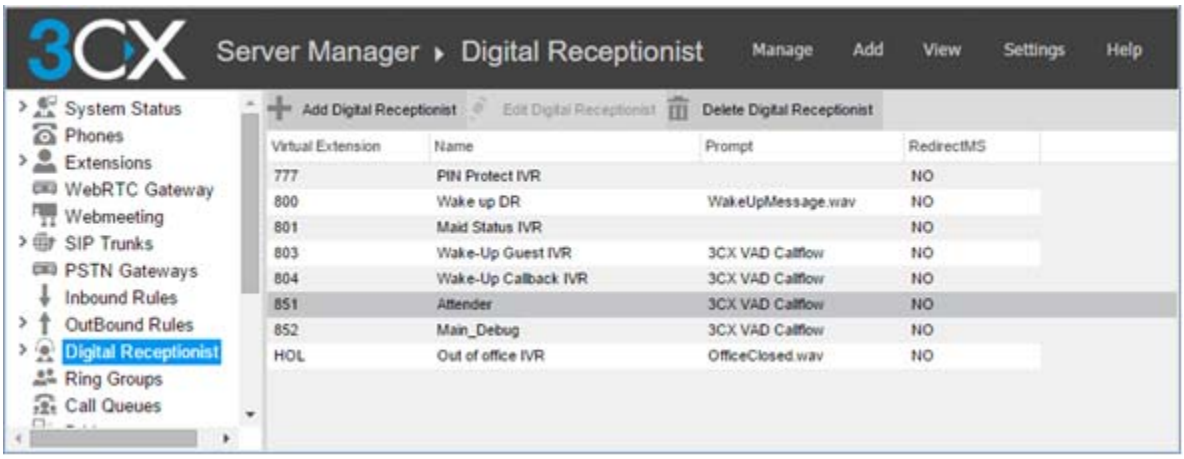
After the deployment has been completed, you will see the callflows registered in 3CX Phone System as Digital Receptionists, and you will be able to manage them as usual.

Executing your voice application

When deploying a project, each callflow is registered in 3CX Phone System with an extension number. The extension number is configured in the ActivationExtension or ActivationExtensionDebug property for the callflow. Be sure not to use an extension number that already exists.

In 3CX Phone System, Voice Application Designer callflows are treated as Digital Receptionists. You can choose when a callflow is activated in the same way you

decide that a Digital Receptionist is activated: for example when an inbound call arrives from a PSTN line, or when a user presses a digit on another Digital Receptionist. After deployment you will see each callflow listed under the Digital Receptionist section.

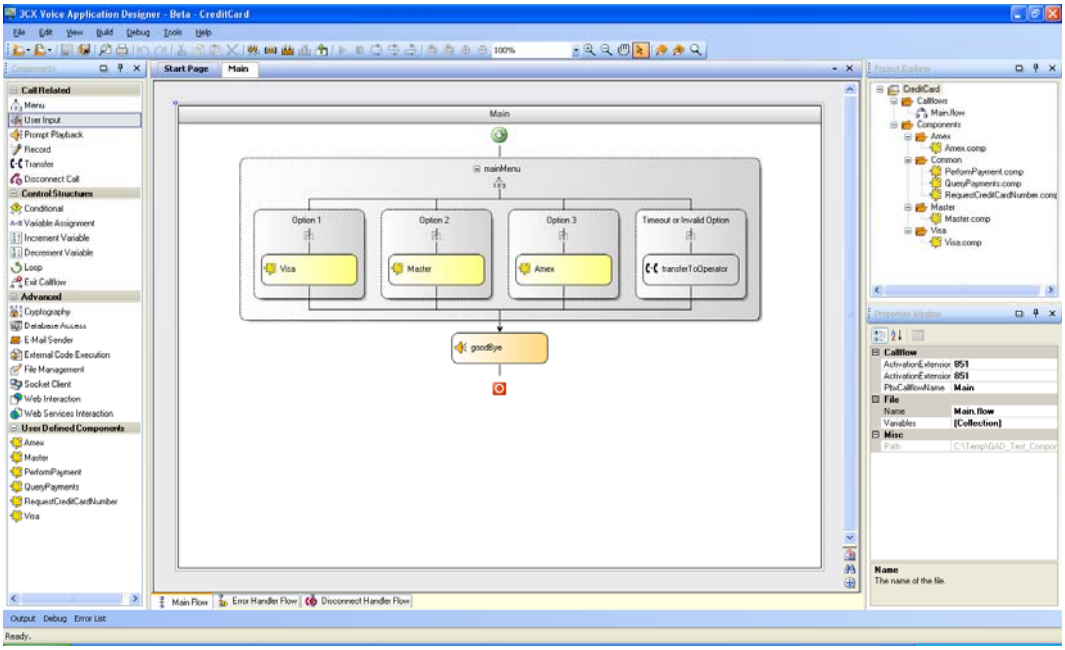


Screenshot 12 - Voice applications are listed as digital receptionists

Now you have built and deployed your voice application you are ready to try it out. Simply ring the extension that you have assigned to the voice application!

3CX Voice Application Designer Workspace

Introduction



Screenshot 13 - The 3CX VAD Workspace

In this chapter we describe the workspace of the 3CX Voice Application Designer. The designer has a complete set of editing functions, including multilevel undo and redo, commands for cutting, copying, pasting and deleting components, commands for enabling, disabling, expanding and collapsing components, and zoom level and navigation tools. It also allows printing flows, previewing the output before printing.

The VAD IDE is composed by the main designer/work space in the centre of the dialog, and a set of docking panels. When a Project is created or opened, a new tabbed document is shown containing the Start Page. The Start Page contains common tasks to be performed on a Project, like opening files, creating callflows or components, or building or deploying the Project. Another tabbed document is shown each time you create a call flow.

The Designer

At the centre of the 3CX VAD is the actual designer, which is shown when you open or create a project or component. The designer consists of one or more tabs with Call flows. A Call flow is a sequence of built in or user defined components. You can drag components from the toolbox and drop them into the designer in the

desired position. Once a component has been placed into the designer, the user can change its properties, using the Properties Window or the configuration form.

There is also a Start page which lists common tasks.

Navigating within the designer

The designer contains buttons below the vertical scroll bar on the right side of the designer, which allow customizing the view:

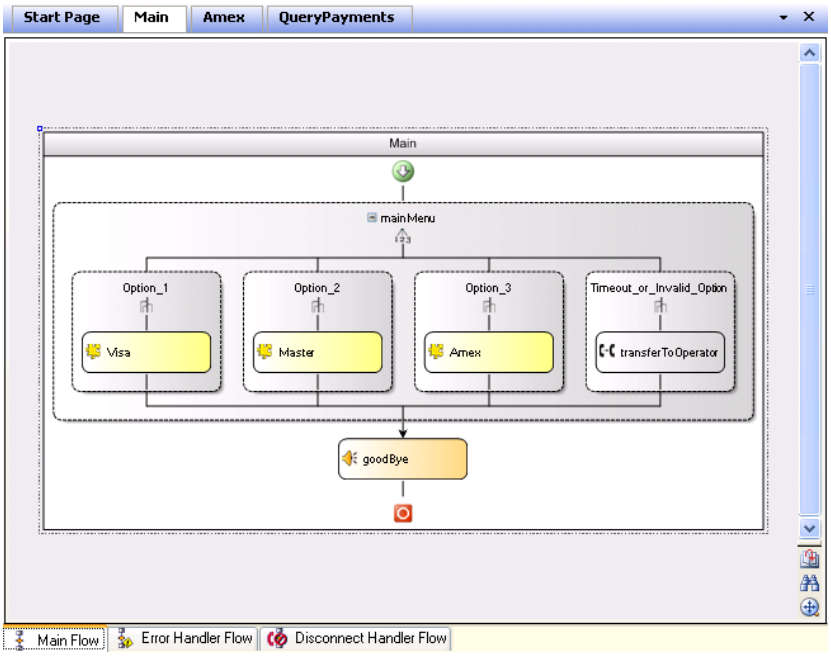
Button	Description
	Fits the designer to the screen size.
	Allows activating or deactivating the Print Preview mode for the current designer.
	Allows changing the zoom level for the current designer.
	Allows to change the pointing device in order to navigate through the designer in the following ways: ■ Zoom in: the Zoom In magnifying glass cursor let the user zoom in by clicking the current designer. ■ Zoom out: the Zoom Out magnifying glass cursor let the user zoom out by clicking the current designer. ■ Navigation tool: the Navigation Tool hand cursor let the user "grab" and shift the view of a flow in the current designer. ■ Default: the Default arrow cursor let the user switch from the other cursors back to the default arrow cursor.

Table 1: Navigation tools within the designer

Types of call flows

There are three kinds of flows:

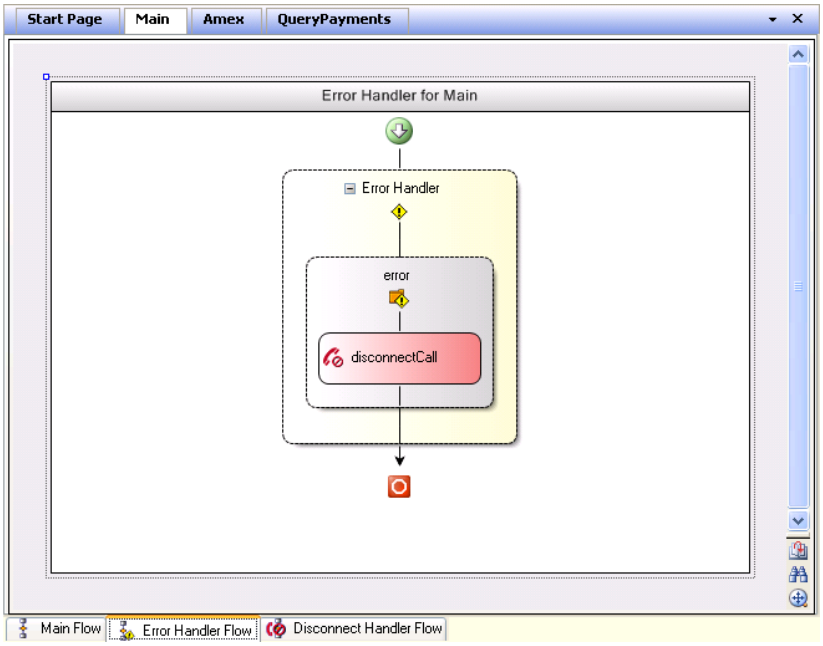
The Main Flow: The execution path in normal conditions. The call will visit each component in the main flow as long as no error occurs. When an error occurs, execution continues goes to the error handler flow. When the call gets disconnected, execution continues on the disconnect handler flow of the document. Call related components cannot be used in the disconnect handler flow.



The Error Handler Flow: if an error occurs in a callflow, this error handler flow ends the execution.

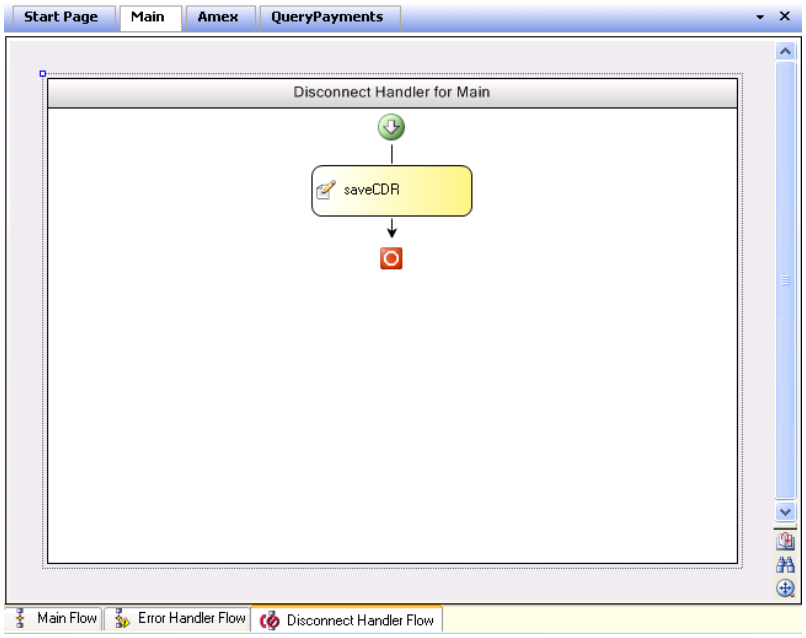
A user defined component is always executed from a callflow or another user defined component containing it. So, when an error occurs while executing a callflow, the error handler of the callflow is executed and then the call is finalized. But when an error occurs while executing a user defined component, the error handler of the user defined component is executed, and then the error handler of its parent is executed, and so on until executing the error handler of the callflow that received the call. It behaves like a stack unwind, executing every error handler upwards, until one handles the error. Then the main flow of the parent is executed.

For example, a call arrives, starts executing Callflow1, which calls Component1, which calls Component2. If an error occurs there, the error handler of Component2 is executed (for example, not handling the error), then the error handler of Component1 is executed (for example, handling the error here), and finally execution continues on the main flow of Callflow1 (because the error was handled on a children component).



The Disconnect Handler Flow: if the call gets disconnected in a callflow, when the disconnect handler flow ends its execution, the whole execution ends. On the other hand, if the call gets disconnected on a user defined component document, when the disconnect handler flow ends its execution, execution continues on the parent document disconnect handler flow.

The behaviour is similar as used for error handling. In this case the call is disconnected. For example, a call arrives, starts executing Callflow1, which calls Component1, which calls Component2. If the connection is disconnected there, the disconnect handler of Component2 is executed, then the disconnect handler of Component1 is executed, and finally the disconnect handler of Callflow1 is executed. The main flow is not executed again for this call.



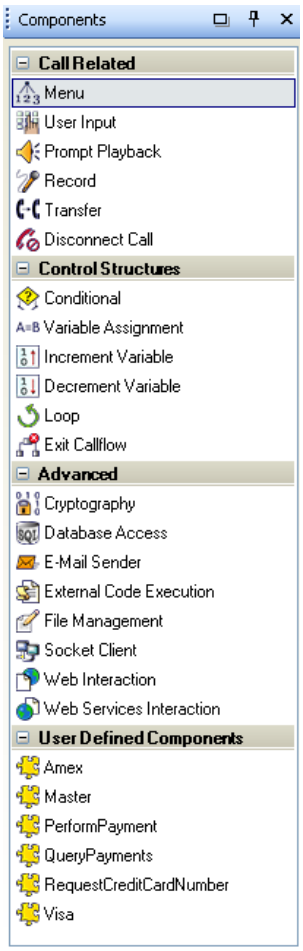
The following table clarifies these concepts:

Event description	Callflow document	User defined component document
Error handled on error handler flow	Whole execution ends after executing the local error handler flow	Execution continues on the parent document main flow after executing the local error handler flow
Error not handled on the error handler flow	Whole execution ends after executing the local error handler flow	Execution continues on the parent document error handler flow after executing the local error handler flow
Connection disconnected	Whole execution ends after executing the local disconnect handler flow	Execution continues on the parent document disconnect handler flow after executing the local disconnect handler flow

Table 2: Error and disconnect handling behavior

The Docking panels

The docking panels contain important functions that allow you to set properties and more for your project.



Screenshot 14 - The components toolbox

Components Toolbox

The toolbox contains a set of built in components. Each user defined component is also added automatically to the toolbox.

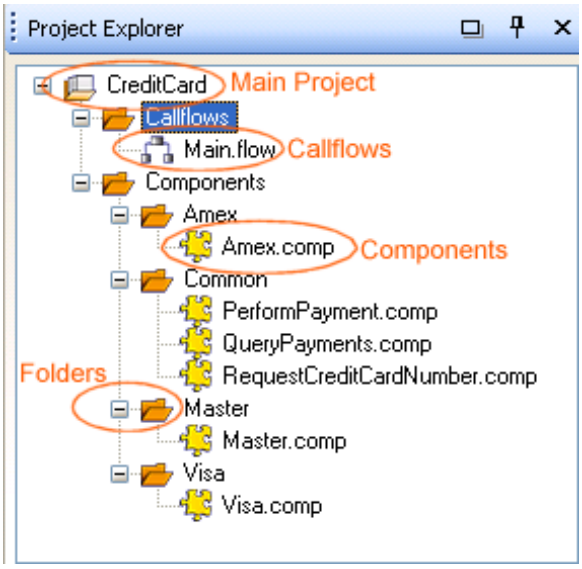
Component	Description
Call Related Components	
Disconnect Call	Disconnects the call.
Menu	Offers a menu with one digit options.
Prompt Playback	Plays back a prompt.
Record	Records audio from the caller.
Transfer	Transfers the call.
User Input	Collects information from the caller.
Control Structures Components	
Conditional	It allows performing actions when a condition is met.
Decrement Variable	It allows decrementing numeric variables.
Exit Callflow	It finalizes callflow execution immediately.
Increment Variable	It allows incrementing numeric variables.
Loop	It allows looping until a condition is met.
Variable Assignment	It allows assigning variables.
Advanced Components	
Cryptography	Encrypts/decrypts data using DES or TripleDES algorithms, and compute MD5 hash.
Database Access	Executes SQL statements on SQL Server, Oracle and ODBC databases.
E-Mail Sender	Sending emails.
External Code Execution	Executes external code, located in libraries or JavaScript source code.
File Management	Allows reading and writing data from file.
Socket Client	Allows establishing of TCP and UDP connections, sending data through the connection and optionally wait for a response from the remote endpoint.
Web Interaction	Allows performing of Web requests.
Web Services Interaction	Allows executing of simple Web Services.
User Defined Components	
User Defined Components	These are components created by the user that are automatically added to the toolbox.

Table 3: Component description

For further information about all the components of the VAD, please refer to Chapter Components.

Project Explorer Window

This docking bar allows you to easily explore the whole Project. Each Project is composed by callflows and user defined components, which can be grouped into folders. The entry point of a call is always a callflow.



Screenshot 15 - The project explorer

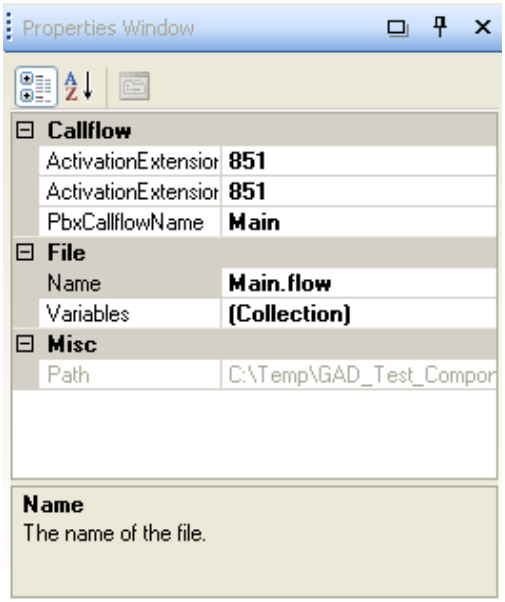
The Project Explorer allows performing the following actions:

- Save the Project or file
- Rename the Project, folder or file
- Close the Project or file
- Create a new folder, callflow or component
- Add an existing callflow or component
- Remove an existing folder, callflow or component
- Build and deploy the Project

Each of those actions can be performed by selecting the option in the context menu, or using shortcut keys. Files and folders can be moved to other locations within the Project using drag and drop. If the VAD cannot find a file or folder referenced by the Project, it is showed in red font, with a hint indicating that the file or folder is missing.

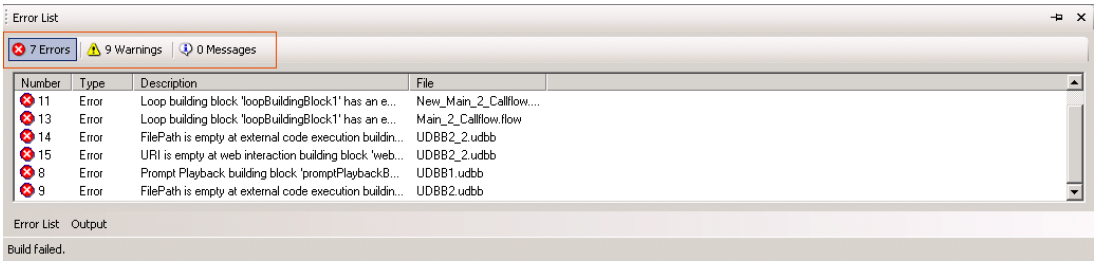
Properties Window

The Properties window shows information of the currently selected object. If the component contains actions, they are shown too, so that they can be quickly executed. For example, the Conditional component contains an “Add Branch” action, which allows adding child conditional branches easily.



Screenshot 16 - The properties window

Error List Window


Error List
7 Errors
9 Warnings
0 Messages
Number Type Description File
11 Error Loop building block 'loopBuildingBlock1' has an e... New_Main_2_Callflow....
13 Error Loop building block 'loopBuildingBlock1' has an e... Main_2_Callflow.flow
14 Error FilePath is empty at external code execution buildin... UDBB2_2.udbb
15 Error URI is empty at web interaction building block 'web... UDBB2_2.udbb
8 Error Prompt Playback building block 'promptPlaybackB... UDBB1.udbb
9 Error FilePath is empty at external code execution buildin... UDBB2.udbb
Error List Output
Build failed.

Screenshot 17 - The Error list window

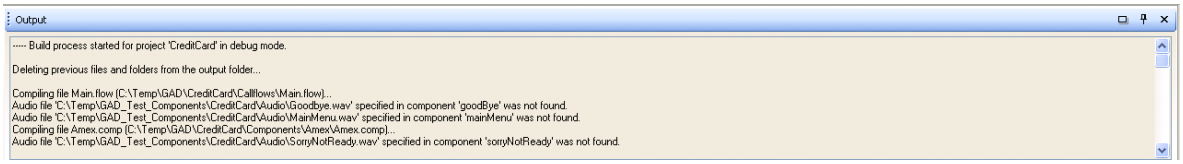
The error list window is used to display messages during the build process. Those messages can be of three types:

- **Errors**: the Project could not be built because of errors in the configuration of one or more of the components. The output could not be generated. The user must correct the errors first.
- **Warnings**: this is something that could be wrong, but the output can be generated. The user should pay attention to this message and verify if he must perform some corrective action.
- **Messages**: this is information that may be considered by the user, but it is not an error and the output can be generated.

Every message includes information related to the component to review, including the file location. Double clicking or pressing <Enter> on a message, the file referred by it is automatically opened.

The VAD offers the possibility to filter the list of messages by pressing the button of the desired message type.

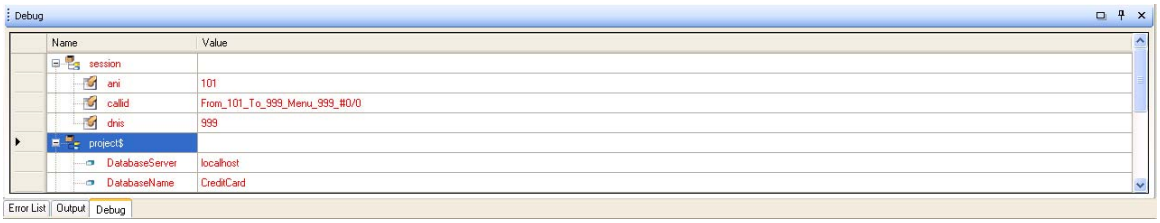
Output Window


Output
----- Build process started for project 'CreditCard' in debug mode.
Deleting previous files and folders from the output folder...
Compiling file Main.flow (C:\Temp\GAD\CreditCard\Callflows\Main.flow)...
Audio file 'C:\Temp\GAD_Test_Components\CreditCard\Audio\Goodbye.wav' specified in component 'goodBye' was not found.
Audio file 'C:\Temp\GAD_Test_Components\CreditCard\Audio\MainMenu.wav' specified in component 'mainMenu' was not found.
Compiling file Amex.comp (C:\Temp\GAD\CreditCard\Components\Vimex\Vimex.comp)...
Audio file 'C:\Temp\GAD_Test_Components\CreditCard\Audio\SorryNotReady.wav' specified in component 'sorryNotReady' was not found.

Screenshot 18 - The Output window

The output window shows the actions performed by the compiler during the build process. For example, the VAD informs when the build process begins, the build mode (Debug or Release), when a file compilation begins, etc.

Debug Window



Screenshot 19 - The Debug window

The debug window shows variable values while a callflow is being debugged. Variables changed during the execution of the last step are shown in red color, while unchanged variables are shown in black color

Components

Introduction

Components are the basic building blocks of a call flow and perform basic actions such as playing of a prompt, recording a call, asking for user input and so on. There are two types of components – one with 'branches', such as the Menu component or without - like the Disconnect Call component.

The Properties window shows input properties for the selected component. Some components have variables which are exposed after execution as output properties. The public properties of user defined components can be used as input and output.

If a red sign appears at the right corner of a component, the component has invalid property values. Click on it to get an explanation.

Prompt Types

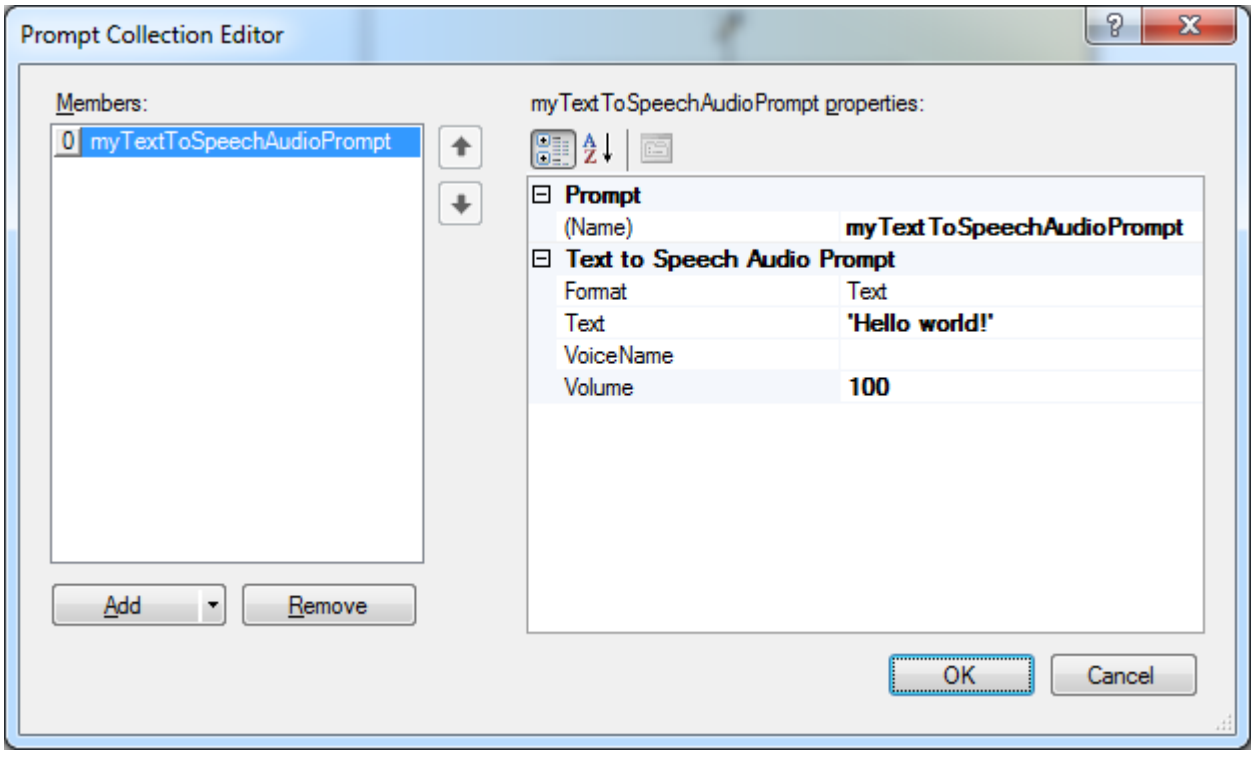
Many VAD components use prompts in order to perform the desired operation. When a component has a prompt collection, each prompt can be of the following types:

- Audio file: The prompt is stored in a wav or mp3 file, and the file name is selected from a file list.
- Dynamic Audio file: The prompt is stored in a wav or mp3 file, and the file name is dynamically created using an expression.
- Recorded Audio: The prompt is stored as a RecordComponent.AudioId variable, containing audio recorded from the end user.
- Text To Speech Audio: The audio is automatically created from text.

When using a Dynamic Audio file prompt, the expression that creates the file name must return a string containing the file path relative to the working directory. You must not use absolute file paths here. If the audio file is located in your project Audio folder, then you just need to refer to it by its name, no special path information is required. For example, if your project is being deployed to %ProgramData%\3CX\Data\Http\Interface\ivr\MyVadProject and your audio file is located at %ProgramData%\3CX\Data\Http\Interface\ivr\ExternalAudio\Audio.wav then the expression must return "..\ExternalAudio\Audio.wav".

When using a Text To Speech Audio Prompt, you need to configure different settings:

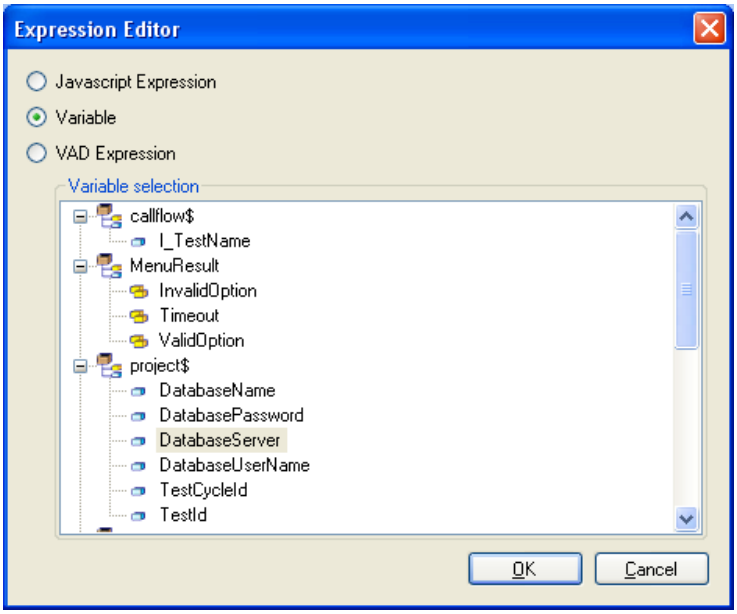
- The Format can be set to "Text" or "SSML".
- The Text property contains the text to convert to speech in one of two different formats. When you set the Format property to "Text", the Text property will be treated as an expression that returns plain text, and that text will be converted to speech. When you set the Format property to "SSML", the Text property will be treated as an expression that returns SSML (Speech Synthesis Markup Language). The essential role of the SSML markup language is to give authors of synthesizable content a standard way to control aspects of speech output such as pronunciation, volume, pitch, rate, etc. For more information read the SSML recommendation: http://www.w3.org/TR/speech-synthesis/
- The VoiceName property allows you to change the default voice used to convert the text to speech. The VAD uses the Microsoft Speech Synthesizer included with the Operating System. If you install additional voices, you can set the voice name here in order to use it. If you need to know what voices you have installed on your server, go to Control Panel → Speech Recognition → Text to Speech. The Voice Selection list will show the available options. By default, "Microsoft Anna" will be available. If you need additional voices, this article explains how to add them: http://www.ehow.com/how_6967571_add-text-speech-voices.html
- The Volume property lets you control the volume of the voice. Valid values are between 0 and 100.
- The Rate property lets you control the audio speaking rate. Valid values are between -10 (extremely slow speech) and 10 (extremely fast speech).



The Text To Speech Audio Prompt is only available on licensed installations.

The Expression Editor

Many input properties can be set using an expression. In this case the input field will contain a icon on the right. Click that button, to open the expression editor.



Screenshot 20 - The expression editor

An expression can be a Javascript Expression, a variable, or a Voice Application Designer Expression.

Variables can be defined at many levels:

- **Project level**: variables defined at this level can be used across the Project by any callflow or user defined component. They are global variables that can be used, for example, to share data between callflows and user defined components.
- **Callflow level**: these variables are visible within the callflow, but cannot be seen by any child component.
- **User defined component level**: these variables can be public or private. Public variables are visible by the parent callflow or component, and can be used to set parameters and customize the component behavior. Private variables are only visible internally at the user defined component.

Variables can be also read-only or read-write.

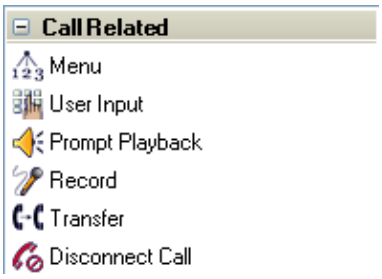
The list of available functions to use in a VAD Expression is as follows:

Function	Description
AND	Performs a logical AND between every parameter, returning a Boolean value as a result. Can have from 2 to 20 parameters.
OR	Performs a logical OR between every parameter, returning a Boolean value as a result. Can have from 2 to 20 parameters.
NOT	Receives a single Boolean parameter and returns another Boolean that is the negation of the provided parameter.
EQUAL	Receives two parameters of any type, and returns a Boolean indicating if they are equal.
NOT_EQUAL	Receives two parameters of any type, and returns a Boolean indicating if they are not equal.
GREAT_THAN	Receives two parameters of any type, and returns a Boolean indicating if the first is greater than the second.
GREAT_THAN_OR_EQUAL	Receives two parameters of any type, and returns a Boolean indicating if the first is greater than or equal to the second.
LESS_THAN	Receives two parameters of any type, and returns a Boolean indicating if the first is less than the second.
LESS_THAN_OR_EQUAL	Receives two parameters of any type, and returns a Boolean indicating if the first is less than or equal to the second.
CONCATENATE	Concatenates every string parameter and returns the resulting string. Can have from 2 to 20 parameters.
LEFT	Receives two parameters. The first is a string to cut. The second is the number of characters to cut. Returns a string that is the first N characters of the original string.
MID	Receives three parameters. The first is a string to cut. The second is the start position where the cut must start, using a zero based index. The third is the number of characters to cut. Returns a string that is the specified substring of the original string.
RIGHT	Receives two parameters. The first is a string to cut. The second is the number of characters to cut. Returns a string that is the last N characters of the original string.
REPLACE	Receives three parameters. The first is a string where the replacement must be made. The second is the text to find in order to replace. The third is the text to replace with. Returns a string with the replacements specified.
REPLACE_REG_EXP	Receives three parameters. The first is a string where the replacement must be made. The second is the regular expression used to find text in order to replace. The third is the text to replace with. Returns a string with the replacements specified.
UPPER	Receives a single string parameter and returns another string that is the original string with upper case.
LOWER	Receives a single string parameter and returns another string that is the original string with lower case.
NOW	Receives no parameters, and returns the current date and time.
LEN	Receives a single string parameter and returns its length as a number.

Function	Description
SUM	Sums every numeric parameter and returns the result. Can have from 2 to 20 parameters.
NEGATIVE	Returns the negative number of the single numeric parameter received.
MULTIPLY	Multiplies every numeric parameter and returns the result. Can have from 2 to 20 parameters.
DIVIDE	Receives two numeric parameters, and returns “first / second”.
ABS	Returns the absolute positive number of the single numeric parameter received.
GET_TABLE_ROW_COUNT	Returns the number of rows of the table received. The table is the result of a database query.
GET_TABLE_CELL_VALUE	Receives three parameters. The first is the variable containing the table (the result of a database query). The second is the row identifier, using a zero based index. The third is the column identifier, using a zero based index. Returns the value of the cell from the specified table, at the specified row and column.

Table 4: Expression Editor – Available functions

Call Related Components



Screenshot 21 - The Call Related Components

Menu

This component configures a single digit menu. It handles the retry logic when the user enters an invalid option or does not enter anything. Its execution finalizes when a valid option is selected or if all retries have been performed.

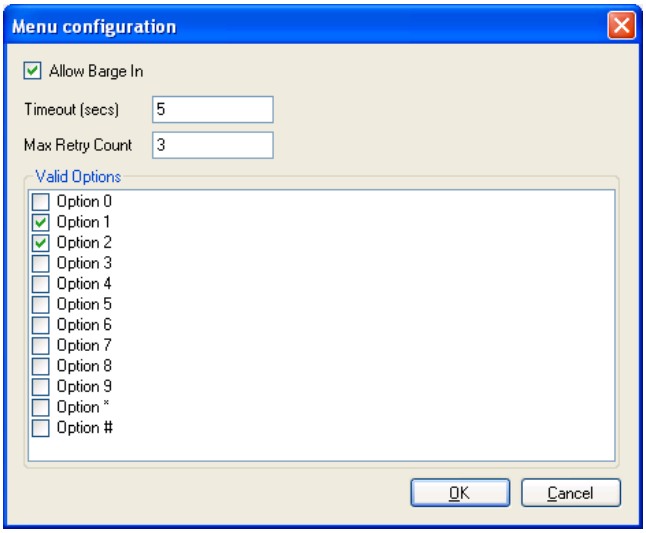
This component has the following input properties:

Property	Description
AllowBargeIn	True to allow the prompts to be barged into. False otherwise.
MaxRetryCount	Number of retry offers for invalid input or timeout. Each one has its own retry counter. For example, setting this property to 5 may allow the user to enter an invalid option 4 times, stay in silence another 4 times, and when the invalid input count or timeout count reaches 5 the menu component will finish its execution.
Timeout	The time to wait for user input before playing the specified timeout prompt, in seconds.

Property	Description
IsValidOption_N	Specify what happens when the user presses N. It could be a valid or an invalid option. Valid options appear as branches in the designer.
InitialPrompts	The list of prompts to play the first time the menu is executed.
SubsequentPrompts	The list of prompts to play subsequent times. The SubsequentPrompts list contains the MinTryCount property and the list of prompts to play when the try count is greater than or equal to MinTryCount. For example, setting MinTryCount = 3 means to use InitialPrompts the first and the second time, and use this prompts after that. You may specify more subsequent prompts each one starting at a different MinTryCount.
TimeoutPrompts	The list of prompts to play when the user does not enter any digit. The same MinTryCount logic applies. After playing the timeout prompt, the main prompt will be played too (initial or subsequent prompt).
InvalidDigitPrompts	The list of prompts to play when the user enters an invalid digit. The same MinTryCount logic applies. After playing the invalid digit prompt, the main prompt will be played too (initial or subsequent prompt).

Table 5: Menu component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 22 - Menu configuration

When the menu ends, execution continues on one of its branches (a valid option branch or the timeout or invalid option branch). The menu component exposes the following output properties for further query:

- **Result**: the result of menu execution. It could be one of the following constant values: MenuResult.Timeout, MenuResult.InvalidOption or MenuResult.ValidOption.

User Input

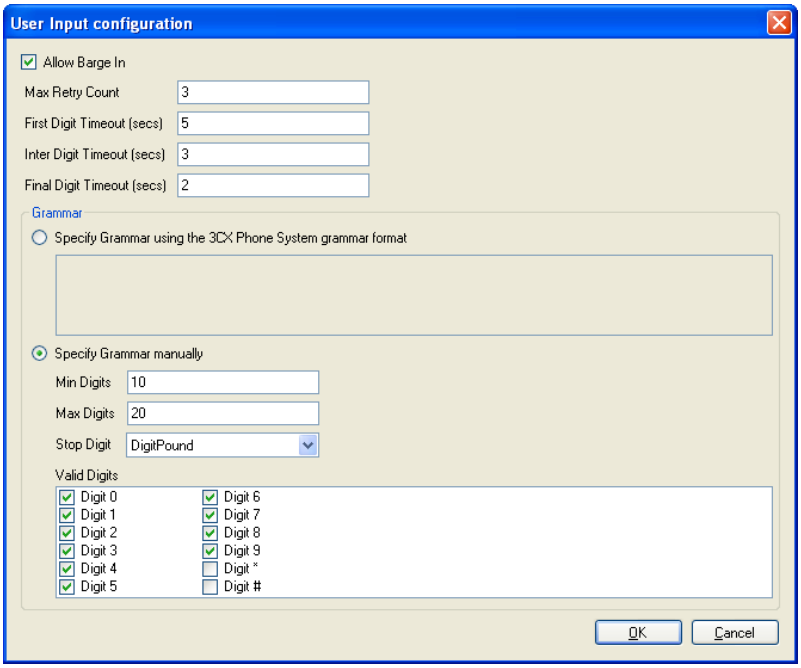
This component collects information from the end user in the form of digits. It handles the retry logic when the user enters an invalid digit, fewer digits than expected or does not

enter anything at all. Its execution finalizes when a valid input is entered by the user or all the retries have been used up.

This component has the following input properties:

Property	Description
AllowBargeIn	True to allow the prompts to be barged into. False otherwise.
MaxRetryCount	Number of retries allowed for timeout or invalid digits input. Each one has its own retry counter. For example, setting this property to 5 may allow the user to enter invalid digits 4 times, stay in silence another 4 times, and when the invalid input count or timeout count reaches 5 the user input component will finish its execution.
FirstDigitTimeout	The time to wait for the first digit before playing the specified timeout prompt, in seconds.
InterDigitTimeout	The time to wait for subsequent digits before playing the specified invalid digits prompt, in seconds.
FinalDigitTimeout	The time to wait for digits after MinDigits has been reached, before returning the entered data, in seconds.
MinDigits	The minimum quantity of digits that must be entered by the user.
MaxDigits	The maximum quantity of digits that can be entered by the user.
StopDigit	Specify the digit that the user must press in order to finalize the data entry.
IsValidDigit_N	Specify what happens when the user presses N. It could be a valid or an invalid digit.
InitialPrompts	The list of prompts to play the first time the user input is executed.
SubsequentPrompts	The list of prompts to play subsequent times. The SubsequentPrompts list contains the MinTryCount property and the list of prompts to play when the try count is greater than or equal to MinTryCount. For example, setting MinTryCount = 3 means to use InitialPrompts the first and the second time, and use this prompts after that. You may specify more subsequent prompts each one starting at a different MinTryCount.
TimeoutPrompts	The list of prompts to play when the user does not enter any digit. The same MinTryCount logic applies. After playing the timeout prompt, the main prompt will be played too (initial or subsequent prompt).
InvalidDigitsPrompts	The list of prompts to play when the user enters invalid digits. The same MinTryCount logic applies. After playing the invalid digits prompt, the main prompt will be played too (initial or subsequent prompt).

Table 6: User Input component – Input Properties



Screenshot 23 - User input configuration

When the user input ends, execution continues on one of its branches ("Valid Input" or "Invalid Input"). The user input component exposes the following output properties for further query:

- **Result**: one of the following values:
- **UserInputResult.Timeout**: the user didn't enter any digit in the last attempt.
- **UserInputResult.InvalidDigits**: the user entered an invalid digit or less than MinDigits in the last attempt.
- **UserInputResult.ValidDigits**: the user entered valid digits between MinDigits and MaxDigits, optionally terminating the input with the StopDigit.
- **Buffer**: the digits entered by the user (including the StopDigit if it was entered by the user).

Prompt Playback

This component allows playing a list of prompts. Each prompt of the list can be of the types described in Prompt Types.

This component has the following input properties:

Property	Description
AllowBargeIn	True to allow the prompts to be barged into. False otherwise.
Prompts	The list of prompts to play when the component is executed.

Table 7: Prompt Playback component – Input Properties

When the prompt playback ends, execution continues to the following component. The prompt playback component does not expose any output properties after its execution.

Record

This component records audio from the caller. Before recording, the list of prompts specified is played to the end user. If the Beep property is set, a beep is played to the user after the prompts and before starting recording.

The recording continues until MaxTime is reached, a silence is detected with FinalSilence length, or the end user presses a DTMF key when the TerminateByDtmf property is set.

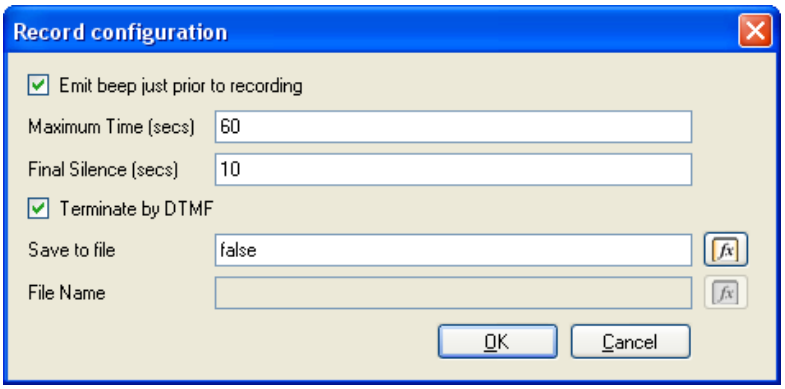
The recorded audio can be automatically saved to a file.

This component has the following input properties:

Property	Description
Beep	If true, a tone is emitted just prior to recording.
Prompts	The list of prompts to play before recording (available prompt types are described in Prompt Types).
MaxTime	The maximum duration to record, in seconds.
FinalSilence	The interval of silence that indicates end of speech, in seconds.
TerminateByDtmf	If true, any DTMF key press will stop the recording.
SaveToFile	If true, the recorded audio will be saved to the file specified by FileName. The value can be an expression.
FileName	The name of the file where the recording must be saved. Only valid when SaveToFile evaluates to true. The value can be an expression.

Table 8: Record component – Input Properties

These input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 24 - Record configuration

The “Save to file” and “File Name” input properties can be set using an expression. In order to create an expression for those properties, press the Expression Editor Button on the right side of the text box. Be aware that in order to enter constant values you need to add quotes. For example, write ‘Some Text’ instead of Some Text.

When the recording ends, execution continues to the next branch (“Audio Recorded” or “Nothing Recorded”). The record component exposes the following output properties for further query:

- **Result**: the recording result:
  - **RecordResult.NothingRecorded**: no audio recorded.
  - **RecordResult.StopDigit**: recording was stopped by a DTMF key press.
  - **RecordResult.StopBySilence**: recording was stopped by a final silence.
  - **RecordResult.Completed**: recording was stopped by reaching the maximum audio length.
- **Duration**: the duration of the recorded audio in seconds.
- **Size**: the size of the recorded audio in bytes.
- **StopDigit**: the DTMF digit pressed by the user when Result = RecordResult.StopDigit.
- **AudioId**: a variable containing the audio, which can be used to play it back to the user.

Transfer

This component allows transferring the call.

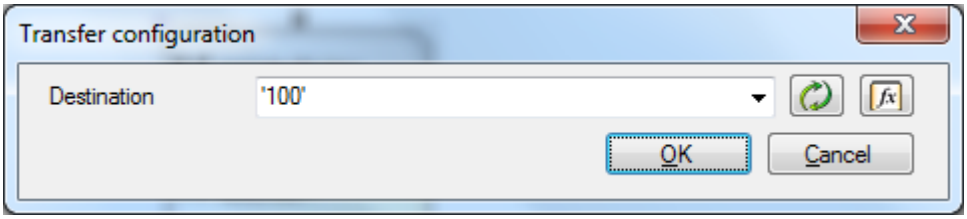
The transfer can be done only in blind mode. The call is automatically disconnected from 3CX Phone System and transferred to the specified destination.

This component has the following input properties:

Property	Description
Destination	The destination number where the call must be transferred. The value can be an expression, so you may for example transfer the call to a number entered by the user in a User Input component.

Table 9: Transfer component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 25 - Transfer component configuration

The “Destination” input property can be set using an expression. In order to create an expression for that property, press the Expression Editor Button on the right side of the text box. Be aware that in order to enter constant values you need to add quotes. For example, write ‘Some Text’ instead of Some Text.

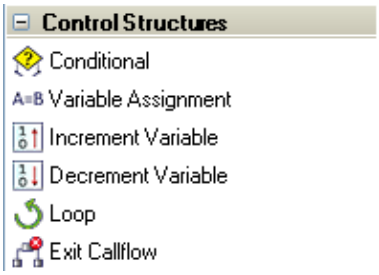
The “Destination” input property can be set choosing an existing extension from 3CX Phone System.

When the transfer ends, execution continues on the disconnect handler flow, as the call is disconnected from the voice application.

Disconnect Call

This component disconnects the current call. It has no special input properties and does not expose any output property after its execution.

Control Structures Components



Screenshot 26 - Control Structure Components

Conditional

This component allows performing an action when a condition is met. The execution path is selected between many branches. This component has no special input properties and does not expose any output property after its execution.

In order to add a branch, the user must execute the “Add Branch” command in the Properties window or the context menu. Each branch has a Condition input property. That property is an expression that must be evaluated to true in order to execute the branch.

Branches are evaluated from left to right, so the user can change the order with the “Move left” and “Move right” commands.

Only one branch can be executed, and it will be the first which has a Condition that evaluates to true. The Condition at the last branch is optional. If it is set, it must be met in order to execute the branch. If it is not set, the branch executes if all previous branches didn’t. If the last branch has a Condition, it is possible that no branch gets executed at all, if every Condition evaluates to false.

Variable Assignment

This component allows assigning a new value to an existing variable. The selected variable must be read-write in order to be assigned.

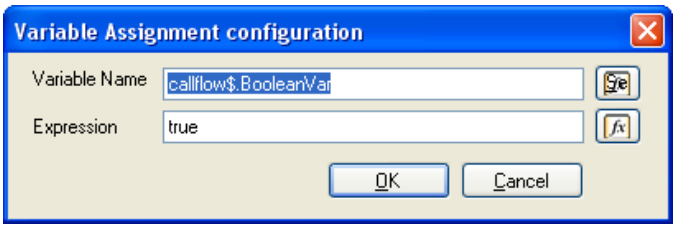
This component has the following input properties:

Property	Description
VariableName	The name of the variable. It must be an existing variable defined at Project scope, callflow scope or a public property of a component.

Property	Description
	The Variable Selector Form allows selecting a variable between all the available ones.
Expression	The value to assign to the variable. The Expression Editor Form allows composing complex expressions more easily. The user can choose to use a Javascript Expression, a variable or a VAD Expression where each parameter is a new expression and can be edited using a new Expression Editor instance. Variables are selected from a list that shows every available variable in the current scope (read-only and read-write variables). For more details about all the available functions, see The Expression Editor.

Table 10: Variable Assignment component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 27 - Variable Assignment configuration form

The “Variable Name” input property can be set using the Variable Selector form. In order to select a variable using the Variable Selector form, press the Browse Button on the right side of the text box.

The “Expression” input property must be set using an expression. In order to create an expression for that property, press the Expression Editor Button on the right side of the text box. Be aware that in order to enter constant values you need to add quotes. For example, write ‘Some Text’ instead of Some Text.

When the variable assignment ends, execution continues on the following component. The variable assignment component does not expose any output property after its execution.

Increment Variable

This component allows incrementing an existing numeric variable (integer, float or currency). The selected variable must be read-write in order to be incremented.

This component has the following input properties:

Property	Description
VariableName	The name of the variable to increment. It must be an existing numeric variable (integer, float or currency) defined at Project scope, callflow scope or a public property of a component. The Variable Selector Form allows selecting a variable between all the available ones.

Table 11: Increment Variable component – Input Properties

When the increment variable ends, execution continues on the following component. The increment variable component does not expose any output property after its execution.

Decrement Variable

This component allows decrementing an existing numeric variable (integer, float or currency). The selected variable must be read-write in order to be decremented.

This component has the following input properties:

Property	Description
VariableName	The name of the variable to decrement. It must be an existing numeric variable (integer, float or currency) defined at Project scope, callflow scope or a public property of a component. The Variable Selector Form allows selecting a variable between all the available ones.

Table 12: Decrement Variable component – Input Properties

When the decrement variable ends, execution continues on the following component. The decrement variable component does not expose any output property after its execution.

Loop

This component allows executing a group of components while a condition is met. The components contained into the Loop component are executed from 0 to N times. If the condition is not met the first time, those components are not executed at all.

This component has the following input properties:

Property	Description
Condition	An expression that must be evaluated to true in order to execute the contained components.

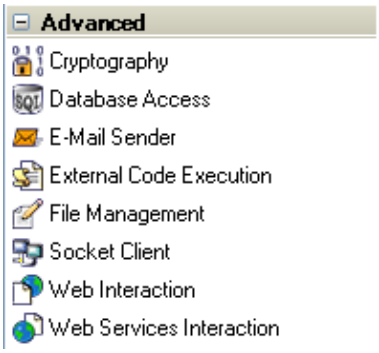
Table 13: Loop component – Input Properties

When the loop ends, execution continues on the following component. The loop component does not expose any output property after its execution.

Exit Application

This component allows exiting the current callflow, also disconnecting the call. It has no special input properties and does not expose any output property after its execution.

Advanced Components



Screenshot 28 - Advanced components

Cryptography

This component allows executing cryptographic algorithms. The supported algorithms are DES and TripleDES, for both encrypting and decrypting, and the MD5 algorithm that performs hash calculation. DES and TripleDES algorithms allow encrypting text using a known key. Then they perform the decryption using the same key.

The hash calculation is a one way algorithm, and the original value cannot be obtained again. The hash calculation is used to validate passwords. The stored information is the hash that was calculated for the original text that was entered by the user, and the validation is performed comparing the hash values.

When the DES algorithm is used, the key must be an 8 characters String. On the other hand, for the TripleDES algorithm, the key must be a 16 or 24 characters String.

The result of the encryption and the MD5 hash calculation is a bytes array. The VAD needs to encode the bytes array in order to store them as a String. There are two possible encoding formats: Hexadecimal and Base64. The hexadecimal format consists of the representation of each byte as two characters that represent the byte value in hexadecimal (from 00 to FF). The Base64 format is used to encode the bytes according to that worldwide known format (http://en.wikipedia.org/wiki/Base64).

The text to be processed can be a constant value, or taken from a variable, or an expression that returns a String.

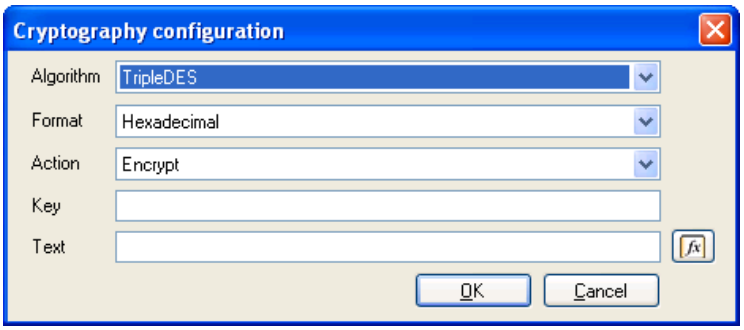
This component has the following input properties:

Property	Description
Algorithm	The algorithm to use. It could be DES, TripleDES or HashMD5.
Format	The encoding format to use in order to transform the encrypted stream into text. When encrypting data, this is how the result is codified. When decrypting data, this is how the input text arrives. It could be Hexadecimal or Base64.
Action	The action to perform. Only valid when Algorithm is DES or TripleDES. It could be Encrypt or Decrypt.
Key	The secret key to be used for the symmetric algorithm. Only valid when Algorithm is DES or TripleDES. It must have a length of 8

Property	Description
	characters for DES, and 16 or 24 characters for TripleDES.
Text	The text to encrypt, decrypt or compute hash. The value can be an expression.

Table 14: Cryptography component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 29 - Cryptography configuration

The “Text” input property can be set using an expression. When the cryptography algorithm ends, flow execution continues on the following component. The cryptography component exposes the following output properties for further query:

- **Result**: stores the processing result encoded according to the specified encryption algorithm or the MD5 hash calculation.

Database Access

This component allows performing operations on SqlServer, Oracle or any other database with ODBC drivers. It allows executing three different kinds of statements: Query, NonQuery and Scalar.

The Query statements return a list of records as result. For example, “SELECT * FROM TABLE”. The NonQuery statements perform an insert or modification, and return the amount of affected records. For example, “INSERT VALUES (VALUE1) INTO TABLE”. The Scalar statements perform data search but return a unique value. For example, “SELECT COUNT(*) FROM TABLE”.

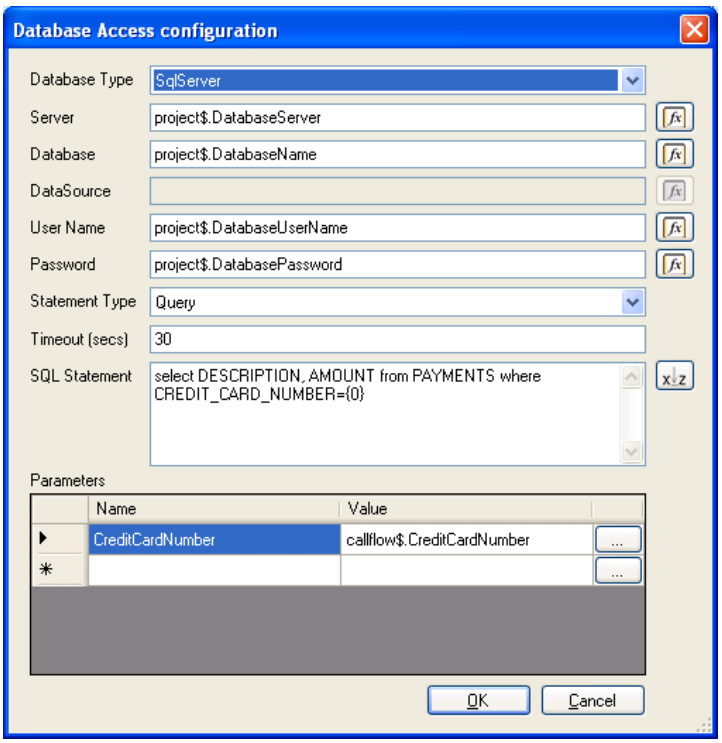
This component has the following input properties:

Property	Description
Server	The database server name or IP address. Only valid when DatabaseType is SqlServer.
Database	The database to use when connecting to the Server. Only valid when DatabaseType is SqlServer.
DataSource	The data source to use when connecting to the database. Only valid when DatabaseType is Oracle or ODBC.
UserName	The username to use when connecting to the database.
Password	The password to use when connecting to the database.

Property	Description
SqlStatement	The SQL statement to execute, including placeholders for variable parameters.
DatabaseType	The type of the database. It could be SqlServer, Oracle or ODBC.
StatementType	The type of the statement. Use Query to execute an SQL statement that returns rows. Use NonQuery to execute an SQL statement that doesn't return rows (for example, INSERT, DELETE, UPDATE, etc.). Use Scalar to execute an SQL statement that returns a single value (for example, SUM, COUNT, etc.).
Timeout	The wait time before terminating the attempt to execute an SQL statement and generating an error. A value of 0 indicates no limit, and should be avoided because an attempt to execute an SQL statement could wait indefinitely.
Parameters	The list of parameters to use with the SQL statement.

Table 15: Database Access component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 30 - Database Access configuration

The “Server”, “Database”, “DataSource”, “User Name” and “Password” input properties can be set using an expression.

The “SQL Statement” input property can contain variable parts. In order to insert a parameter value into the SQL statement, press the button on the right side of the text box and select the desired parameter. Parameters can also be set using an expression. In order to create an expression for a parameter, press the ellipsis button on the last column of the grid.

When the database access ends, execution continues on the following component. The database access component exposes the following output properties for further query, which contain the processing result according to the type of the selected statement:

- **QueryResult**: a table containing every row returned from the database. The expression editor functions GET_TABLE_ROW_COUNT and GET_TABLE_CELL_VALUE allow scanning the whole table. The first function returns the amount of records in the table, so it can be stepped through with the Loop component. The second function allows obtaining the cell value specifying the row and column numbers. Indexes start at zero.
- **NonQueryResult**: an integer value containing the number of rows affected.
- **ScalarResult**: a string value containing the value returned from the database.

E-Mail Sender

This component allows sending of e-mail messages.

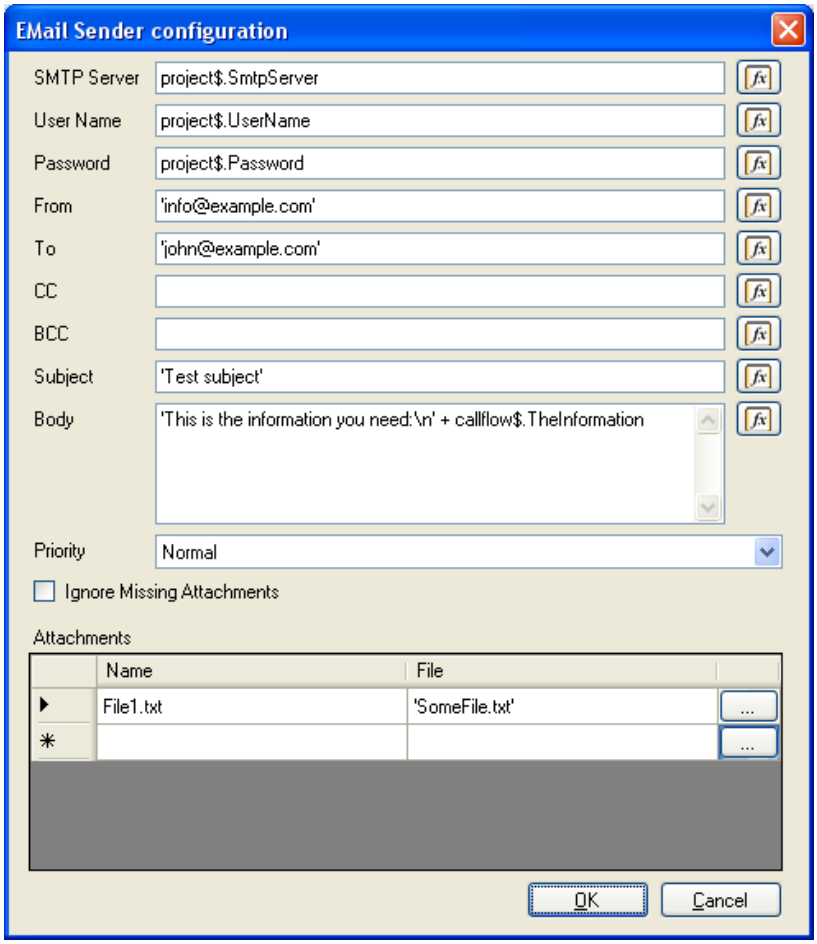
This component allows using authenticated connections with user name and password, specifying the sender and different addresses types (To, CC or BCC) determining the subject and mail body, and attaching files.

This component has the following input properties:

Property	Description
UserName	The username to use when connecting to the SMTP Server.
Password	The password to use when connecting to the SMTP Server.
Server	The SMTP Server name or IP address.
EnableSSL	A Boolean value to specify if the connection to the SMTP Server must be done using SSL or not.
From	The e-mail address to use in the 'from' field of the message.
To	The e-mail address to use in the 'to' field of the message.
CC	The e-mail address to use in the 'cc' field of the message.
BCC	The e-mail address to use in the 'bcc' field of the message.
Subject	The subject of the e-mail message.
Body	The body of the e-mail message.
Priority	The priority of the e-mail message.
IgnoreMissingAttachments	True to silently ignore missing attachment files on runtime, False to cause a runtime error when that condition occurs.
Attachments	The list of attachments to send with the e-mail message.

Table 16: E-Mail Sender component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 31 - Email configuration

The “SMTP Server”, “EnableSSL”, “User Name”, “Password”, “From”, “To”, “CC”, “BCC”, “Subject” and “Body” input properties can be set using an expression. The name of the attachment to send is a constant string value (does not require quotes). Attachment files can be determined using an expression. In order to select the expression for the attachment file in the server, press the ellipsis button on the last column of the grid.

When the e-mail sender ends, execution continues on the following component. The e-mail sender component does not expose any output property after its execution.

External Code Execution

This component allows executing code located in a JavaScript file, a COM component, or a .NET library. When used to execute code in a JavaScript file, the user must specify the name of the function to call and the list of parameters to use.

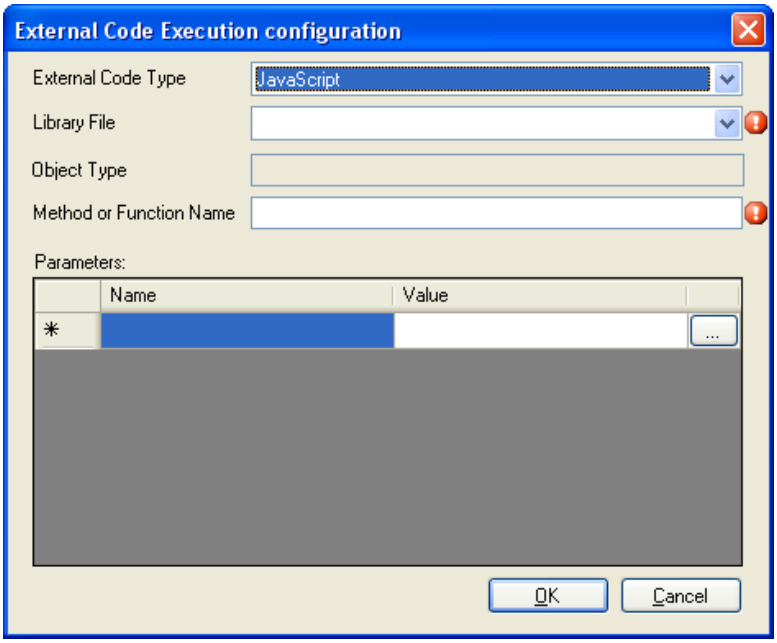
When used to execute code in a COM (ActiveX) component or .NET library, the VAD creates an instance of the specified object and invokes the specified method on that instance, using the list of parameters provided.

This component has the following input properties:

Property	Description
ExternalCodeType	The type of the external code to execute: JavaScript, COM or DotNetLibrary.
FilePath	The path to the file or library to execute. Only valid when ExternalCodeType is JavaScript or DotNetLibrary.
ObjectType	The type of the object to create. Only valid when ExternalCodeType is DotNetLibrary or COM. When working on a .NET library, this is the class name including the namespace. When working on a COM component, this is the ProgID.
MethodName	The name of the method or plain function to execute. When ExternalCodeType is DotNetLibrary or COM, an object of type ObjectType will be created and the method will be executed on that object instance.
Parameters	A list of parameters used when invoking the function or method. Each parameter can be a constant value, a variable or an expression. The name of each parameter is informative, and is not used to validate against the function or method to call.

Table 17: External Code Execution component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 32 - External Code Execution configuration

The Library File is selected from a drop down list. The list contains every library file or JavaScript document in the “Libraries” folder of the project.

Parameter names are constant string values and do not require quotes. Parameter values can be set using an expression. In order to create an expression for a parameter, press the ellipsis button on the last column of the grid. Be aware that in order to enter constant values you need to add quotes. For example, write ‘Some Text’ instead of Some Text.

When the external code execution ends, execution continues on the following component. The external code execution component exposes the following output properties for further query:

- **ReturnValue**: contains the value returned by the function or method called.

File Management

This component allows reading or writing text files. The reading operation is performed line by line. It is possible to specify the index of the first line to be read (starting at zero), and the amount of lines to be read. It also can be set that all the lines are read until reaching the end of the file.

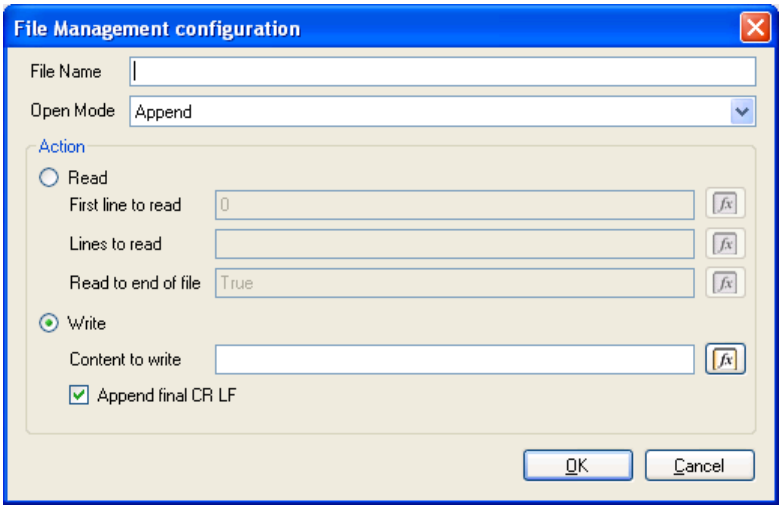
The writing operation allows opening the file in three modes: “Append”, “Create” and “Open”. The “Append” mode allows adding text at the end of the file. The “Create” mode allows creating the file if it does not exist, or truncate it if it already exists, so it always starts with an empty file. The “Open” mode allows opening the file as it is, and write to it since the first character, overriding what was on it. The data to be written is the information that is stored in the “Content” property. If the AppendFinalCrLf property is set, Carriage Return and Line Feed characters are added at the end of the line.

This component has the following input properties:

Property	Description
OpenMode	Use Append to append data at the end of the file when Action is Write. Use Create to open a new file or truncate an existing one. Use Open to open an existing file.
Action	The action to perform. It could be Read or Write.
FileName	The path (absolute or relative from the deployment web server), to the file to read or write.
Content	The text to write to the file. Only valid when Action is Write.
AppendFinalCrLf	True to append a final CR LF after writing to the file. False otherwise. Only valid when Action is Write.
LinesToRead	The number of lines to read from the file. Only valid when Action is Read.
FirstLineToRead	The zero-based index of the first line to read from the file. Only valid when Action is Read.
ReadToEnd	Overrides LinesToRead, reading until the end of file is reached. Only valid when Action is Read.

Table 18: File Management component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window. The configuration form for the File Management component is showed in the following figure.



Screenshot 33 - File Management configuration

The “First line to read”, “Lines to read”, “Read to end of file” and “Content to write” input properties can be set using an expression.

When the file management ends, execution continues on the following component. The file management component exposes the following output properties for further query:

- **EOF_Reached**: indicates it the end of the file has been reached when Action is Read.
- **Result**: contains the text that has been read from the file when Action is Read.

Socket Client

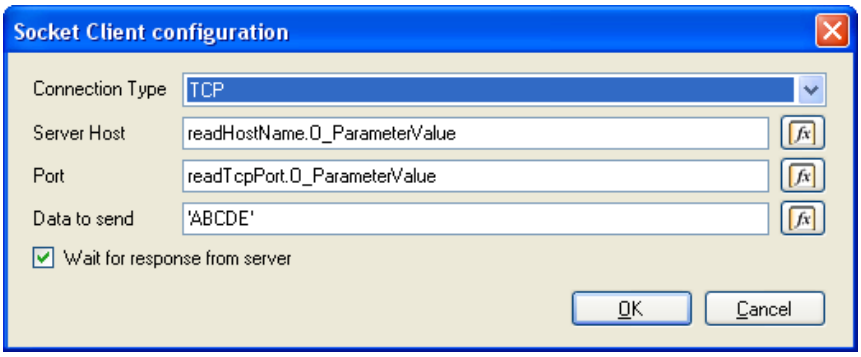
This component allows sending and receiving data using TCP or UDP protocols. The destination server and port are specified, with the information to be sent. If the property WaitForResponse is set, it waits for a response from the server, otherwise it closes the socket immediately after the information is sent.

This component has the following input properties:

Property	Description
ConnectionType	The type of the connection to establish. TCP or UDP.
Host	The server host name or IP address.
Port	The port number where the server is listening for incoming connections.
Data	The data to send to the server.
WaitForResponse	True to wait for a response after sending the data. False otherwise.

Table 19: Socket Client component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 34 - Socket Client configuration

The “Server Host”, “Port” and “Data to send” input properties can be set using an expression. When the socket client ends, execution continues on the following component. The socket client component exposes the following output properties for further query:

- **Response**: contains the server response if the WaitForResponse property was set.

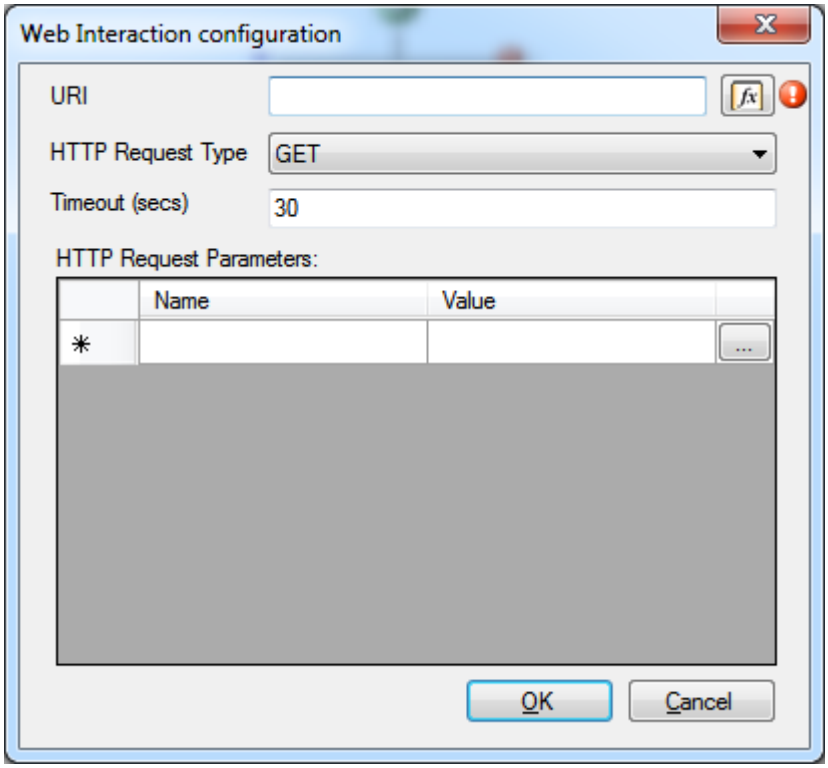
Web Interaction

This component simulates a Web browser, requesting a web page to a server. It allows using the GET or POST methods, by encoding the specified parameters as it corresponds.

This component has the following input properties:

Property	Description
URI	The URI where the request must be sent.
HttpRequestType	The type of the HTTP Request. Available values are “GET” and “POST”.
Timeout	The time to wait while trying to connect to the Web Server before returning the Server Timeout condition, in seconds. Zero means to wait forever.
Parameters	A list of parameters used when performing the request. Each parameter can be a constant value, a variable or an expression. The name of each parameter is the name of the field that will be sent to the web server (query string or form data).

Table 20: Web Interaction component – Input Properties



Screenshot 35 - Web interaction configuration

The “URI” input property can be set using an expression. HTTP Request Parameter names are constant string values and do not require quotes. HTTP Request Parameter values can be set using an expression.

When the web interaction ends, execution continues on the following component. The web interaction component exposes the following output properties for further query:

- **HttpStatusCode**: Contains the HTTP status code returned by the web server. The number 200 (OK) means that the request was processed successfully.
- **Content**: When HttpStatusCode = 200 this property contains the response from the web server as a string.

In previous versions the URI property was a constant string value, and now it has been changed to be an expression, in order to give it more flexibility. When the VAD opens a file saved with a previous version, it will automatically convert the constant value to an expression. After saving the project with this version of the VAD, you will not be able to open it again with previous versions.

Web Services Interaction

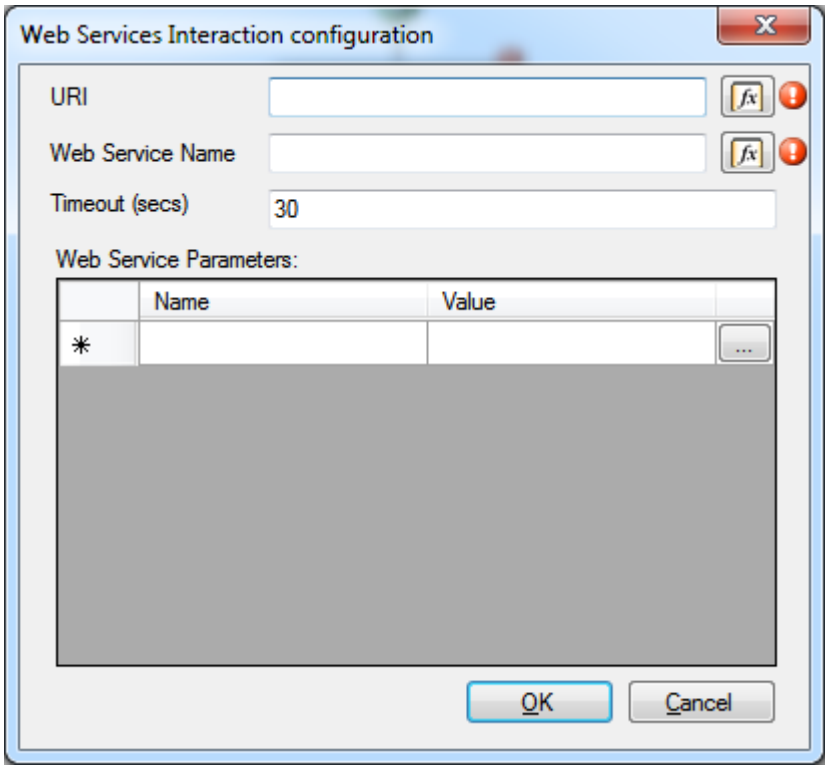
This component allows executing simple xml Web services. It performs a web request using the POST method, to the specified URI and WebServiceName (for example: http://www.example.com/ExampleWebServiceName), using the provided parameters.

This component has the following input properties:

Property	Description
URI	The URI where the web service is located.
WebServiceName	The name of the method to invoke.
Timeout	The time to wait while trying to connect to the Web Service before returning the Server Timeout condition, in seconds. Zero means to wait forever.
Parameters	The list of parameters to send when invoking the web method. Each parameter can be a constant value, a variable or an expression. The name of each parameter is the name of the field that will be sent to the web service as form data.

Table 21: Web Services Interaction component – Input Properties

Those input properties can be set using the configuration form, activated from the context menu at the designer or the link in the properties window.



Screenshot 36 - Web Services Interaction configuration

The “URI” and “Web Service Name” input properties can be set using an expression. Web Service Parameter names are constant string values and do not require quotes. Web Service Parameter values can be set using an expression.

When the web services interaction ends, execution continues on the following component. The web services interaction component exposes the following output properties for further query:

- **Result**: contains the root element content extracted from the XML returned by the web service. If the return value is a complex XML, this property will contain the complex XML, except the removed root element. In simple xml web services, the answer is usually a plane value into the root element of the response (this component is designed to be used when executing simple xml web services).

In previous versions the URI and WebServiceName properties were constant string values, and now they have been changed to be expressions, in order to give them more flexibility. When the VAD opens a file saved with a previous version, it will automatically convert the constant value to an expression. After saving the project with this version of the VAD, you will not be able to open it again with previous versions.

If you're creating the web service using the .NET framework, and need to access it remotely (for example, 3CX Phone System running on computer A and the web service running on computer B), then you will need the explicitly enable the HTTP POST protocol in the "web.config" file. This can be done adding the following settings:

```
<configuration>
    <system.web>
        <webServices>
            <protocols>
                <add name="HttpPost"/>
            </protocols>
        </webServices>
    </system.web>
</configuration>
```

User Defined Components

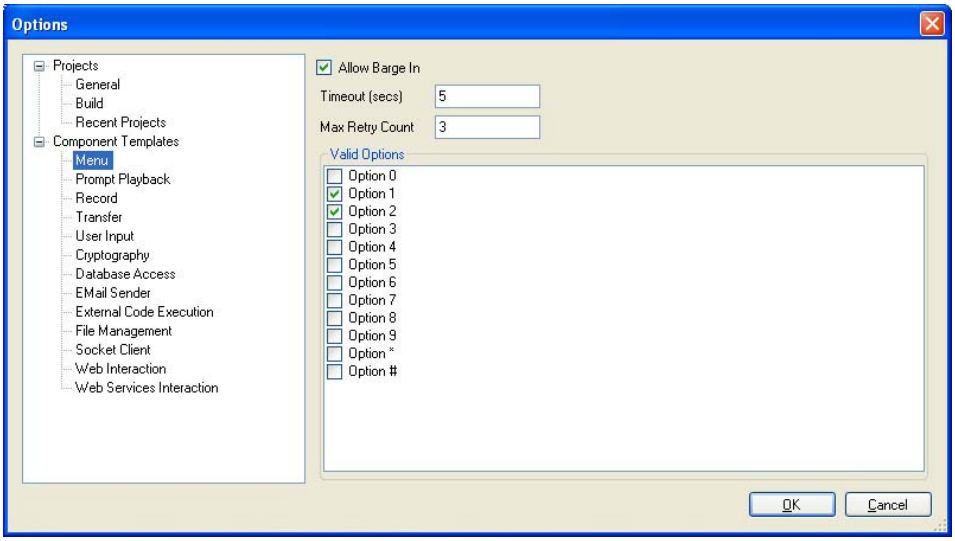
User defined components are automatically added to the toolbox. They can be used to group common functionality, reduce diagram complexity and as a localized error or disconnect handler.

Public properties exported by these components are automatically shown in the Properties window. The user can assign an expression to those properties in order to customize the component behaviour. Be aware that in order to enter constant values you need to add quotes. For example, write 'Some Text' instead of Some Text.

The component can internally update properties values so they can be used as component output too.

Default Component options

You can change the default options of the various components from the Tools > Options menu. The configured values will be used when dragging a component from the toolbox and dropping it into the designer.



Screenshot 37 - Component options

Menu

- **Allow Barge In**: Set it to True to allow the prompts to be barged into. Set it to False otherwise.
- **Timeout (secs)**: The time to wait for user input before playing a timeout prompt, in seconds.
- **Max Retry Count**: the quantity of retry offers for invalid input or timeout.
- **Valid Options**: the list of valid options that are automatically added to the menu component when created.

Prompt Playback

- **Allow Barge In**: set it to True to allow the prompts to be barged into.

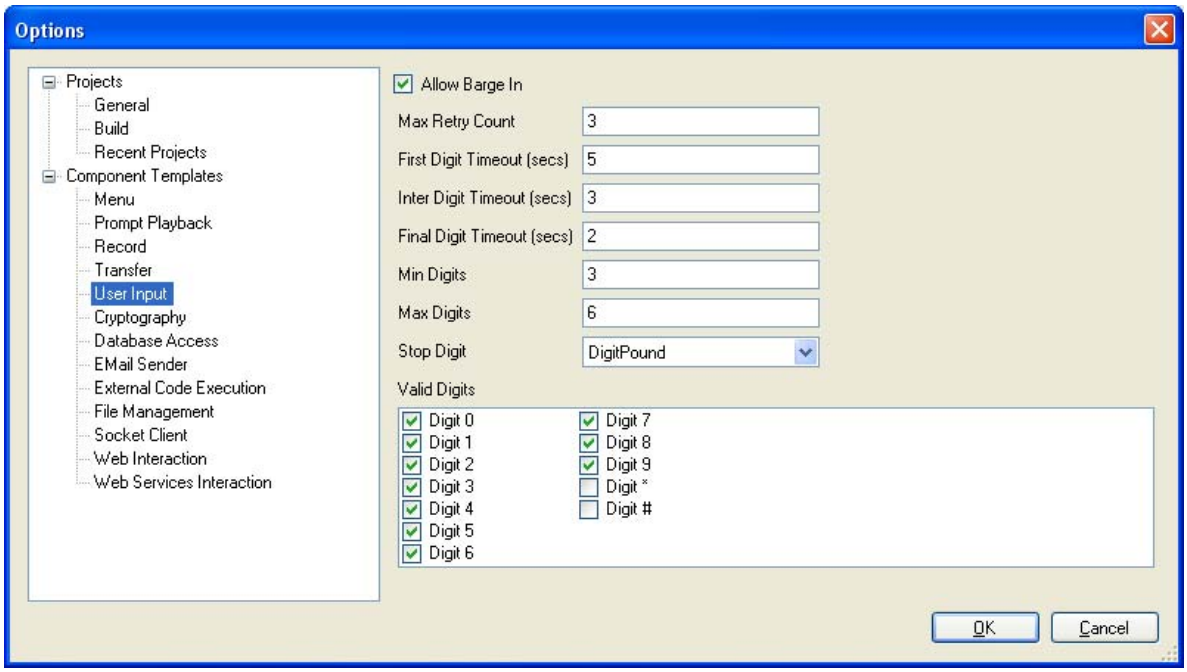
Record:

- **Emit beep just prior to recording**: it indicates if a beep sound is played before recording.
- **Maximum Time (secs)**: the maximum duration to record, in seconds.
- **Final Silence (secs)**: the interval of silence that indicates end of speech, in seconds.
- **Terminate by DTMF**: if true, any DTMF key will stop the recording.
- **Save to file**: if true, the recorded audio will be saved to file.

Transfer

- **Automatically load extensions from 3CX Phone System when configuration starts**: Set to true to automatically load extensions information from the 3CX Phone System when the configuration form is opened.

User Input



Screenshot 38 - Options – Component Templates – User Input

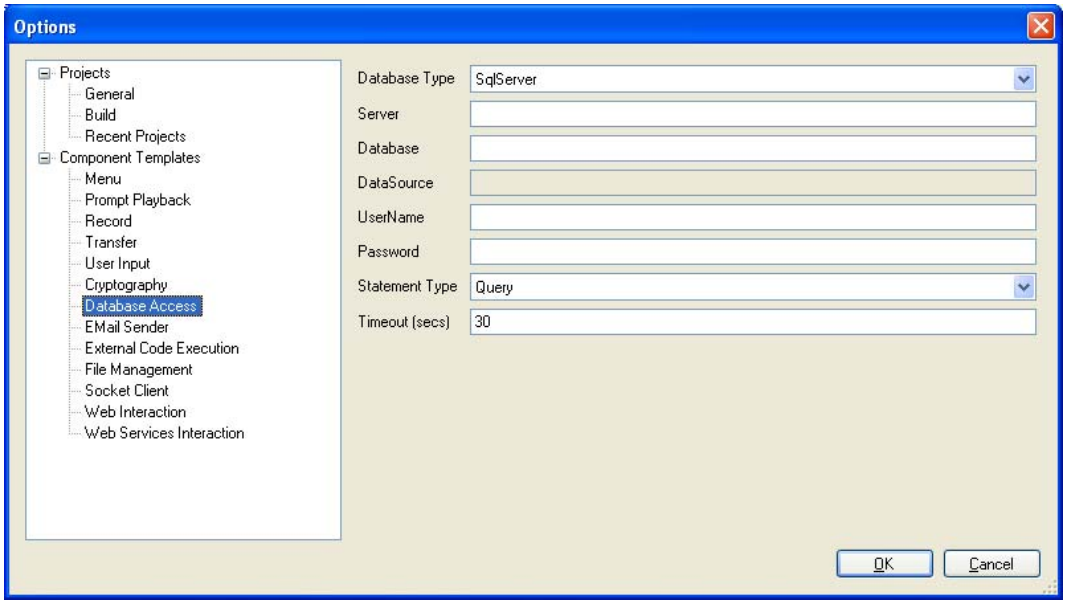
- **Allow Barge In**: set it to True to allow the prompts to be barged into.
- **Max Retry Count**: the number of retries allowed.
- **First Digit Timeout (secs)**: the time to wait for the first digit before playing a timeout prompt, in seconds, or zero to use the default value.
- **Inter Digit Timeout (secs)**: the time to wait for subsequent digits before playing an invalid digits prompt, in seconds, or zero to use the default value.
- **Final Digit Timeout (secs)**: the time to wait for digits after MinDigits has been reached, before returning the entered data, in seconds, or zero to use the default value.
- **Min Digits**: the minimum number of digits that must be entered by the user.
- **Max Digits**: the maximum number of digits that can be entered by the user.
- **Stop Digit**: specify the digit that the user must press in order to finalize the data entry.
- **Valid Digits**: indicate which digits will be considered as valid when they are entered by the user.

Cryptography

- **Algorithm**: the algorithm to use (DES, TripleDES or MD5 hash).
- **Format**: encoding format for encrypted or hash bytes.

- **Key**: the key to use with DES and TripleDES algorithms.

Database Access:



Screenshot 39 - Options – Component Templates – Database Access

- **Database Type**: the type of the database (SQL Server, Oracle or ODBC).
- **Server**: the database server name or IP address. Only valid when Database Type is SQL Server.
- **Database**: the name of the database to use when connecting to the database server. Only valid when Database Type is SQL Server.
- **Datasource**: the datasource to use when connecting to the database server. Only valid when Database Type is Oracle or ODBC.
- **UserName**: the user name to use when connecting to the database.
- **Password**: the password to use when connecting to the database.
- **Statement Type**: the type of the statement (Query, NonQuery or Scalar).
- **Timeout (secs)**: the wait time before terminating the attempt to execute an SQL statement and generating an error. Zero means to wait forever.

E-Mail Sender:

- **SMTP Server**: the SMTP server name or IP address.
- **EnableSSL**: true to connect to the SMTP server over SSL, false otherwise.
- **UserName**: the user name to use when connecting to the SMTP server.
- **Password**: the password to use when connecting to the SMTP server.
- **From**: the e-mail address to use in the “from” field of the message.

- **To**: the e-mail address to use in the “to” field of the message. It can be a comma separated list.
- **CC**: the e-mail address to use in the “cc” field of the message. It can be a comma separated list.
- **BCC**: the e-mail address to use in the “bcc” field of the message. It can be a comma separated list.
- **Subject**: the subject of the e-mail message.
- **Body**: the body of the e-mail message.
- **Priority**: the priority of the e-mail message.
- **Ignore Missing Attachments**: True to silently ignore missing attachment files on runtime, False to cause a runtime error when that condition occurs.

External Code Execution:

- **External Code Type**: the type of the external code to execute (JavaScript, DotNetLibrary or COM).

File Management:

- **Open Mode**: the mode in which the file must be opened. Use “Append” to append data at the end of the file when Action is “Write”. Use “Create” to open a new file or truncate an existing one. Use “Open” to open an existing file.
- **Action**: the action to perform (Read or Write).

Socket Client:

- **Connection Type**: the type of the connection to establish (TCP or UDP).
- **Server Host**: the server host name or IP address.
- **Port**: the port number where the server is listening for incoming connections.
- **Wait for response from server**: indicates if the component needs to wait for a response from the server after sending the data.

Web Interaction:

- **HTTP Request Type**: the type of the HTTP request (GET or POST)
- **Timeout (secs)**: The time to wait while trying to connect to the Web Server before returning the Server Timeout condition, in seconds. Zero means to wait forever.

Web Services Interaction

- **Timeout (secs)**: The time to wait while trying to connect to the Web Service before returning the Server Timeout condition, in seconds. Zero means to wait forever.

Debugging

Introduction

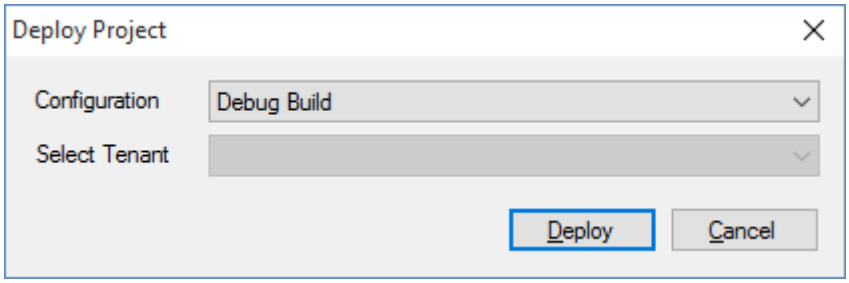
The VAD allows debugging of callflows, following the execution path graphically and watching variable values changes step by step. When a call activates a callflow built in debug mode, a debug information file is generated containing variable values and the execution path. That file can be opened later by the VAD in order to debug the callflow.

Building in Debug mode

In order to debug a callflow, you must build the project in debug mode. After getting a successful debug build, you must deploy the debug configuration to 3CX Phone System. Now every call processed by the project's callflows will generate a debug information file, which can be imported latter by the VAD in order to graphically show you the execution path and variable values. In this chapter we will use the CreditCard sample application as an example. We will make a call and import the debug information file into the VAD in order to debug the callflow.

To build the Project in Debug mode:

1. Open the Project.
2. From the main window of the 3CX Voice Application Designer, choose the command Build → Debug Build.
3. Once you got a successful debug build, you can execute the main window menu command Build → Deploy.

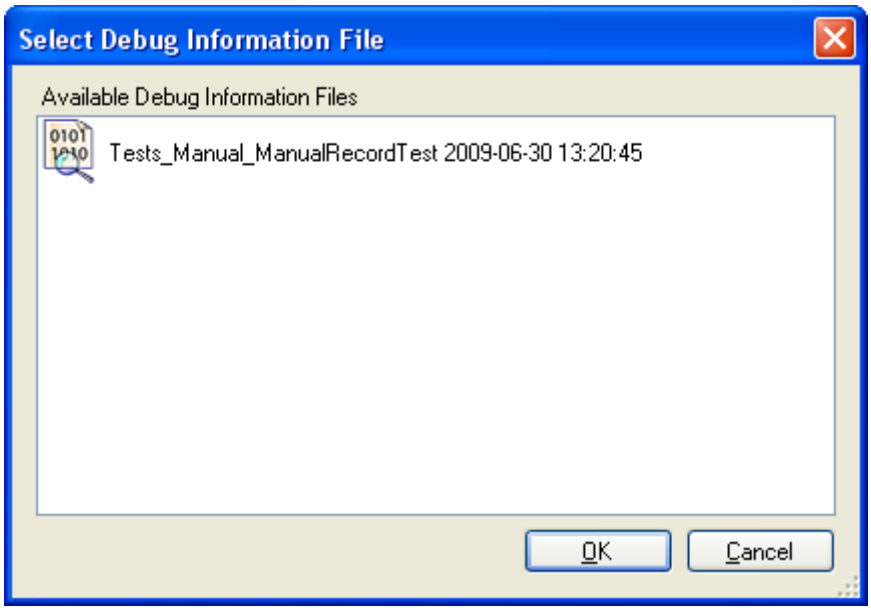


4. The Deploy Project dialog will appear. Select 'Debug Build'. Then click Deploy. We are ready to make the call and start debugging the callflow!

Callflow debugging

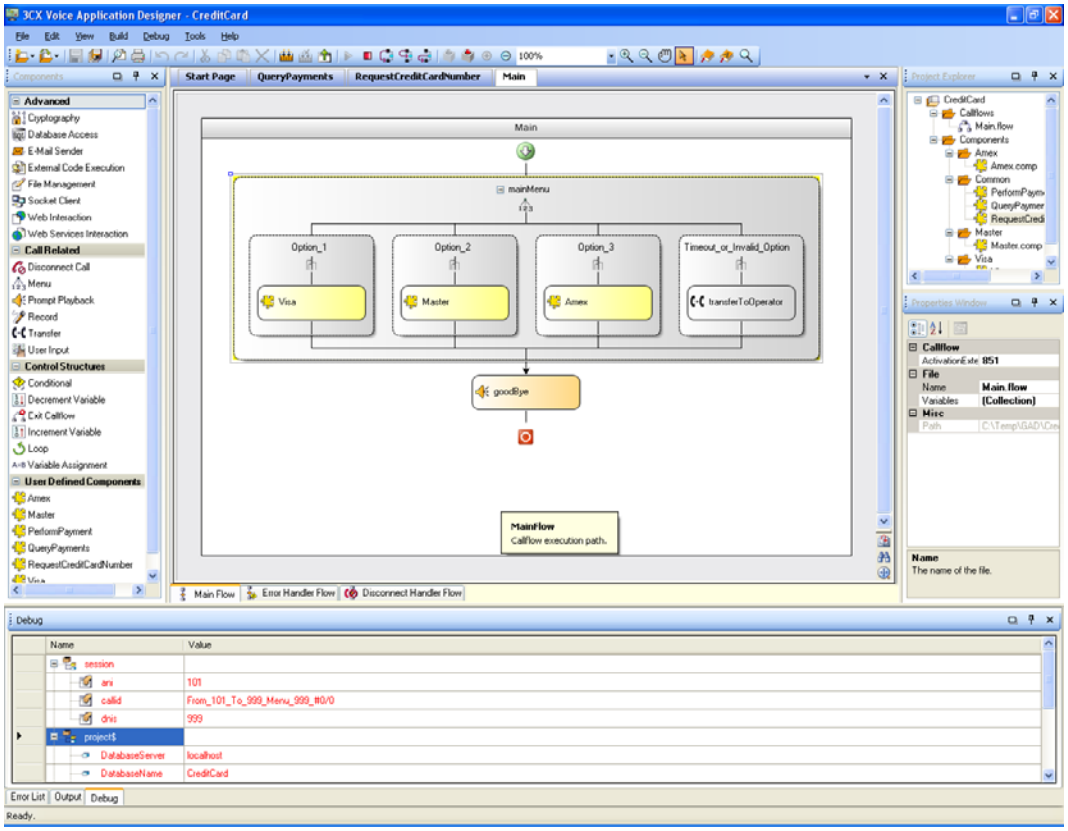
To debug a call, you need to call the voice application so that it will be executed in debug mode. Then follow the next steps:

1. From the main window toolbar, press the Start Debugging Button .

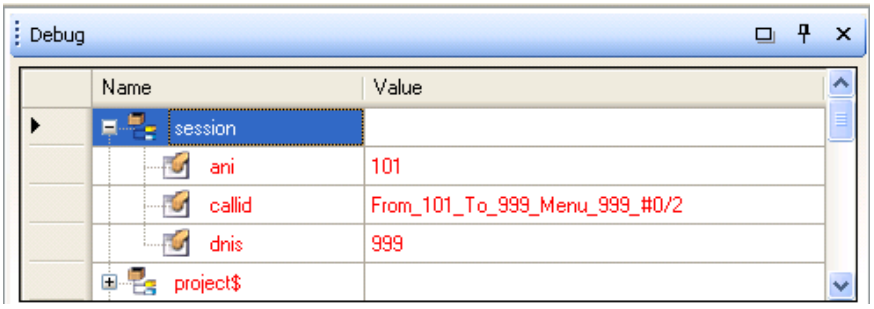


2. The debug information file will be displayed with the date and time the call was started, and the callflow executed. Select the desired debug information file and click OK.

3. The VAD loads the file information and starts debugging the callflow, highlighting the first executed component with a yellow background, and showing the initial variable values in the Debug Window.

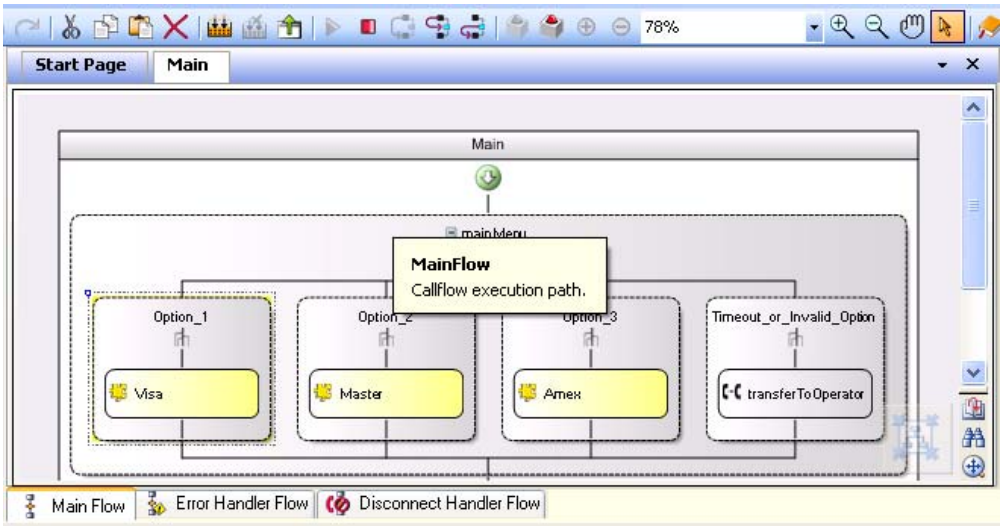


4. Variables updated during the last step are shown in red color. Initially all variables are showed in red color because that is the first assigned value.



5. The available debug command buttons are enabled automatically in the main menu and the main toolbar after the execution of each step.

6. Pressing the “Step Into” Button , the callflow execution continues to the next component. This menu command is available when the currently executing component is a User Defined Component or a component with child branches. For example, when you are executing a menu component:



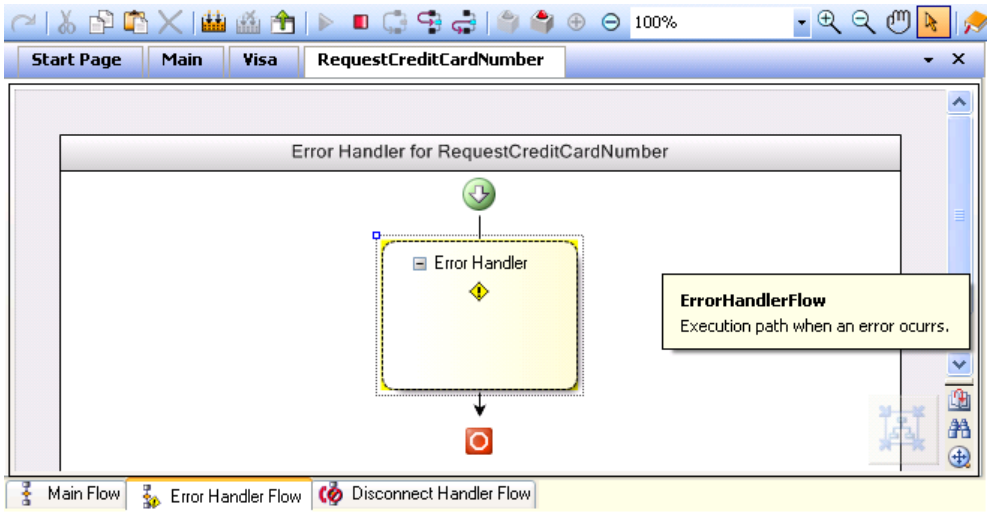
Screenshot 40 - Step into command

7. Pressing the “Step Over” Button , will cause the callflow execution to continue to the next component at the same level as the current component. This menu command is available when the currently executing component is not the last one in the designer.

8. Pressing the “Step Out” Button , the callflow execution continues in the next component in the parent component level. For example, pressing the Step Out Button when the currently executing component is a menu branch, the execution continues at the component located after the parent menu component. The “Step Out” Button is enabled until the last executed component is reached.

9 .Pressing the “Stop Debugging” Button , the debugging process is stopped.

10. If any error occurs, the callflow execution continues in the Error Handler Flow.



11. If the call is disconnected, the callflow execution continues in the Disconnect Handler Flow.

Sample Application

Introduction

In this chapter we are going to create a voice application and go through all the steps involved in making the application. As an example, we are going to create a simple Credit Card system, allowing querying and performing payments. We will show how to reuse components to reduce complexity. We will also show how error handling works and how to configure it properly. The following figure describes the Callflow diagram that we will create:

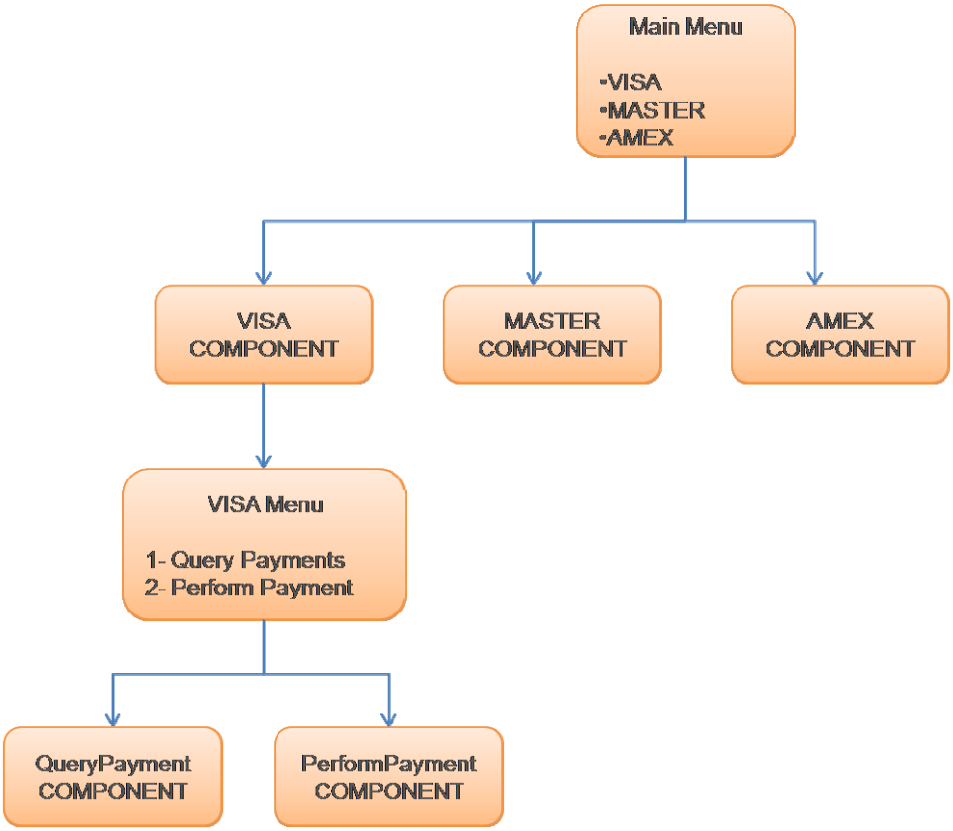


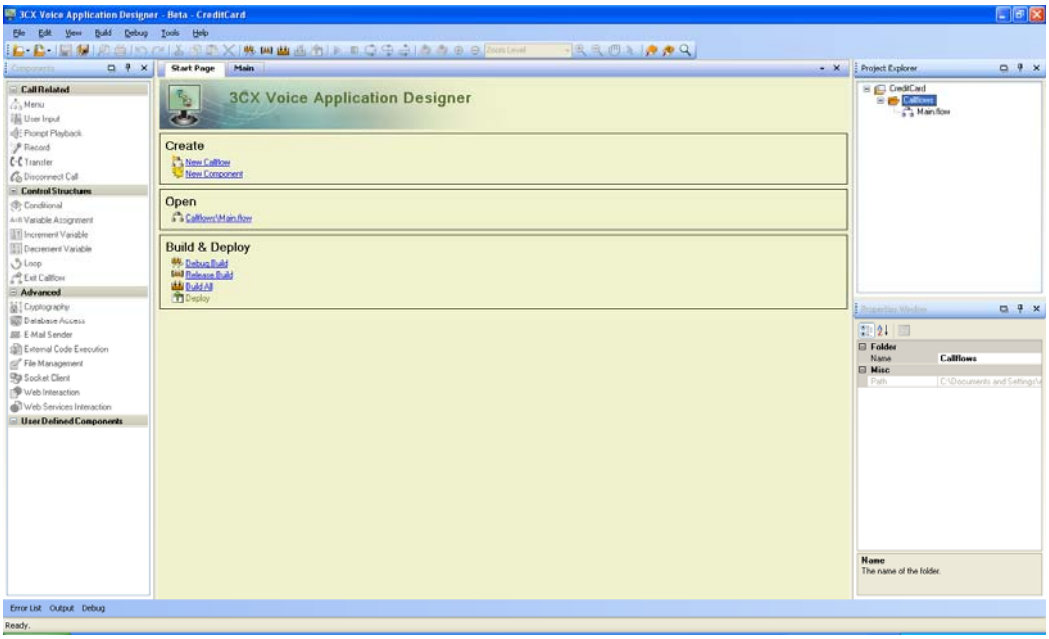
Figure 1: Example - Callflow diagram

Creating and configuring the Callflow

Creation of the Project and the "Main" Callflow

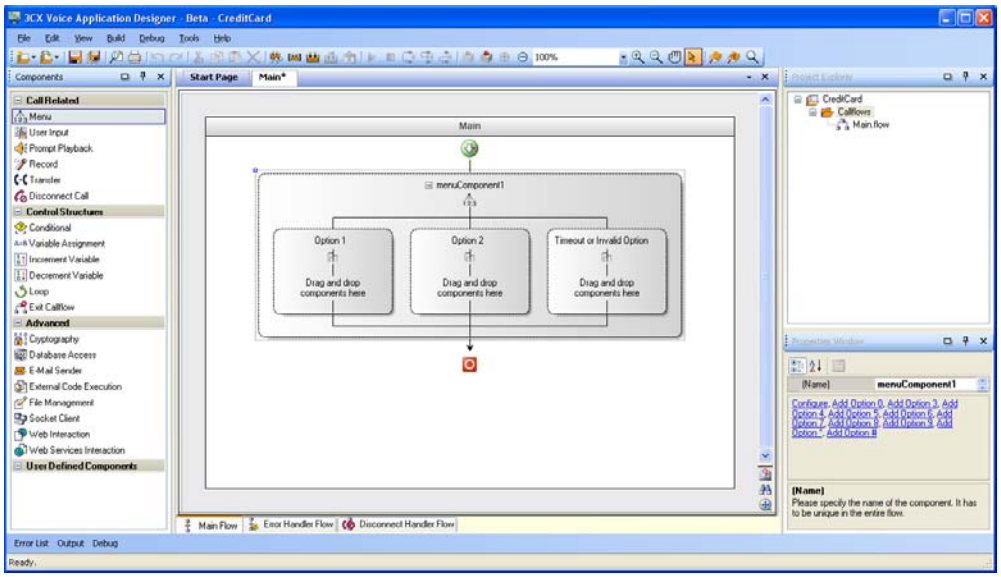
1. File → New Project - Select the folder where you want to create the Project and enter the name: **Credit Card**

2. Press Save in order to create the Project
3. We are now going to create a new Folder within the Project, where all the Callflows will be stored. From the Project Explorer, right click on the name of the Project (CreditCard). Choose New Folder, and name it “Callflows”:



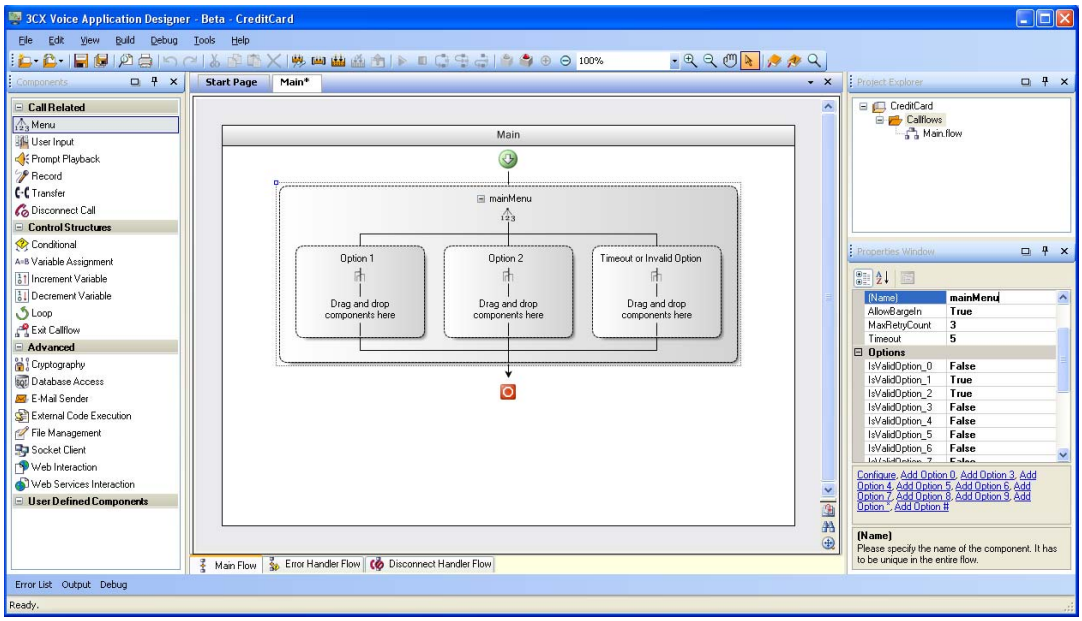
Screenshot 41 - Creating a new folder for the call flows

4. Now, we are going to move the automatically created “Main” callflow into the recently created folder. To do this, drag it and drop it into that folder.
5. We need to add the main menu to the callflow. Double-click on the callflow file. From the Call Related Components section, click and drag the Menu component, and drop it at the beginning of the Main Callflow.



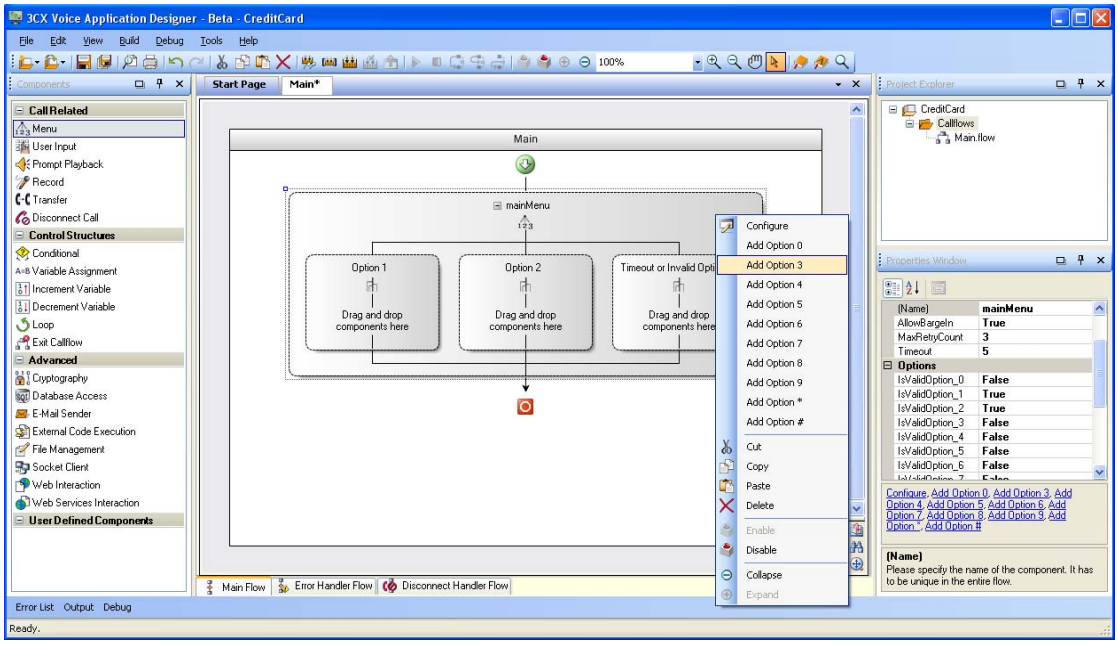
Screenshot 42 - Adding a menu component

6. Now we will modify the name of the added menu component. Select the menuComponent1 component and from the Properties windows modify the "(name)" property with **mainMenu**:



Screenshot 43 - Modifying the name

7. We need a main menu with 3 options (1 Visa, 2 Master, 3 Amex), so we are going to add a new option. To perform this operation, right click on the mainMenu component and execute the "Add Option 3" menu command from the context menu:

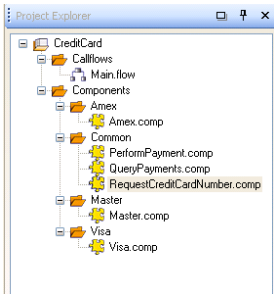


Screenshot 44 - Adding a menu option

8. Before going on with the “Main” callflow, we are going to create some folders that will let us store the components. Using the command to create a New Folder (right click on the Project Explorer and choose New Folder), lets create the following folders inside de CreditCard Project: CreditCard -> Callflows- > Components Amex, Master, Visa, Common
9. Now we are ready to create the User Components that will implement each one of the main menu options.

Configuring the Main Callflow

Now we must create the User Components.



Screenshot 45 - Creating the user components

1. From the Project Explorer:

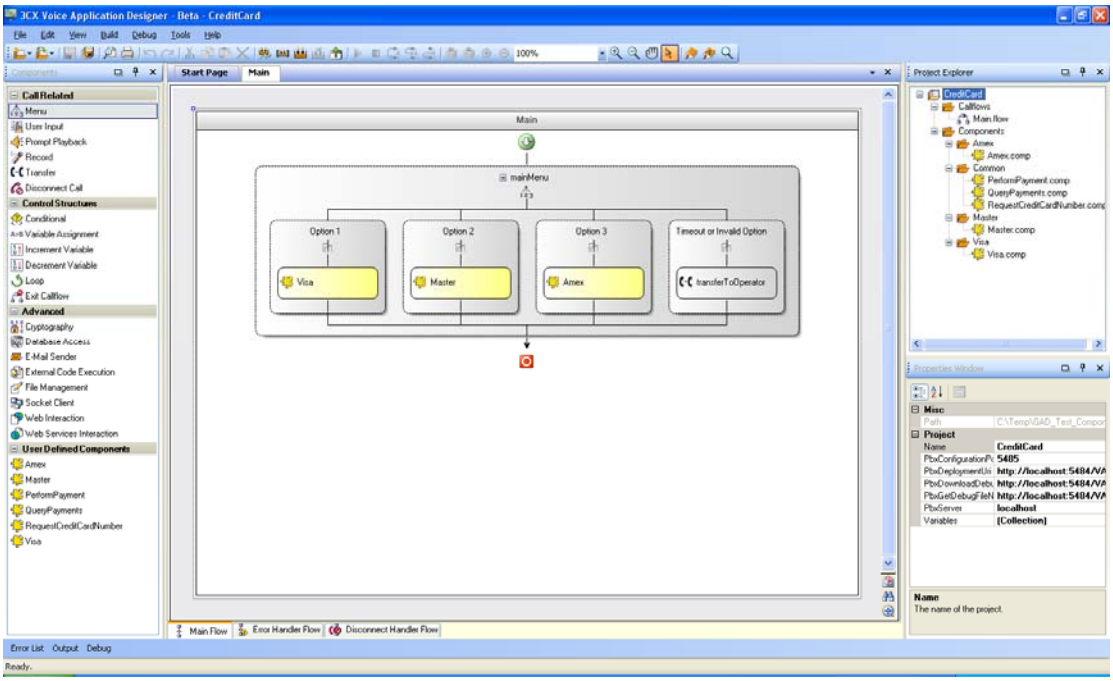
- Right click on the Amex folder and choose New Component. Name it **Amex.comp**.
- Repeat for the Master folder and name it **Master.comp**.
- Again for the Visa folder and choose New Component. Name it **Visa.comp**.
- Right click on the Common folder and choose New Component. Create the following components: **PerformPayment.comp**, **QueryPayments.comp** and **RequestCreditCardNumber.comp**.

2. Now we are going to add the components to the main menu options of the Main callflow:

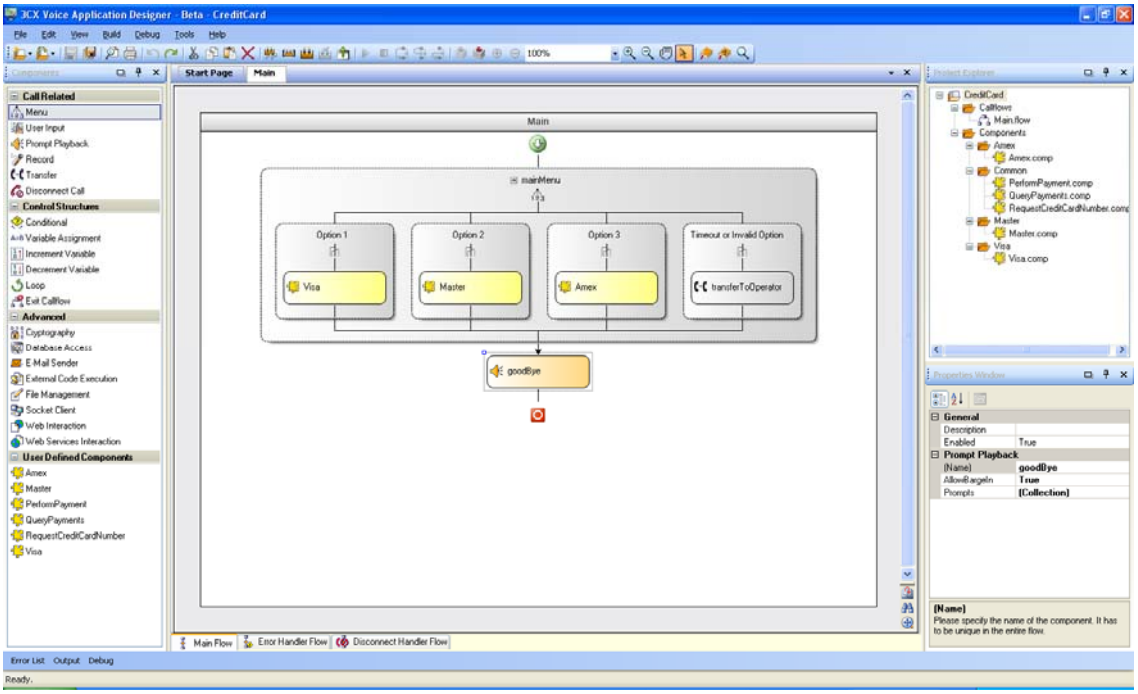
- From the User Defined Components Section, choose the Visa component that corresponds to the first menu option, and drag it into the Option_1. Then, from the Properties Window, select the User Component recently added, and change the component name with Visa.
- Repeat this to insert Master and Amex components into Option_2 and Option_3 respectively. Remember to modify their names from the Properties Window.

3. Now we are going to configure the Timeout or Invalid option branch. From the Call Related Components section, drag the Transfer Component and drop it into the Timeout or Invalid option branch. From the Properties window, select Transfer

(name) and type transferToOperator. The following figure shows the resulting callflow:



4. Now we are going to add a message to be played at the end of the Main callflow. From the Call Related Components section, drag the Prompt Payback Component to the Main callflow, and drop it at the end of the callflow. From the Properties window, modify the component name to **goodBye**. The following figure shows the resulting callflow:



Screenshot 46 - The main call flow

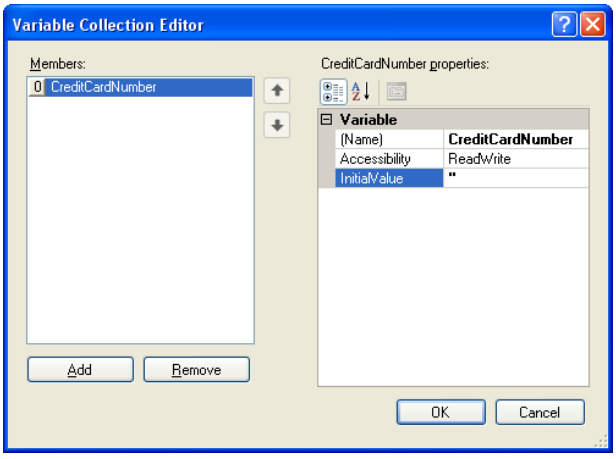
The “Main” callflow is ready. Now we need to design the created User Components.

Remember that you may move callflow and component files by dragging and dropping them in the new location in the Project Explorer.

Configuring User Components variables

Now we are going to configure the user component’s variables.

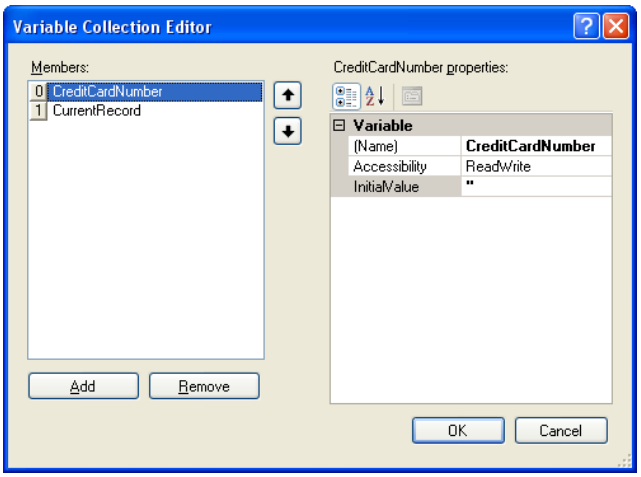
1. We will begin with RequestCreditCardNumber component. This component will ask for a credit card number and will be used by Visa, Master and Amex components. It needs to return the credit card number entered by the end user, so we need a public variable to store it.



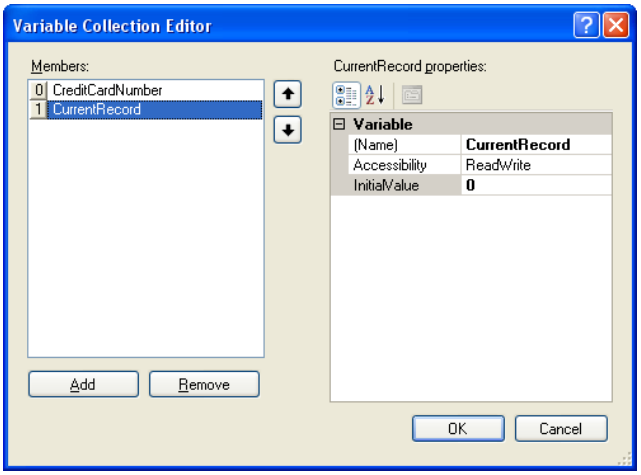
Screenshot 47 - Variables

Click on the Variables button, from the Properties window. Click the Add button. Configure it as shown.

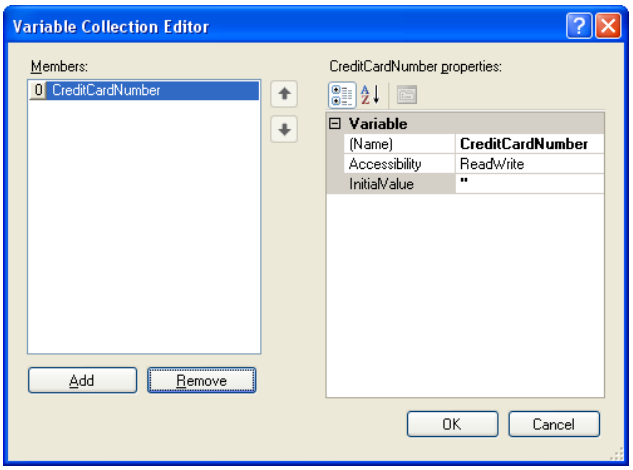
2. Now we are going to configure the QueryPayments component. This component will query a database in order to get a list of payments. The CurrentRecord variable will allow us to loop through that list. Configure the following variables:



and



3. It's time for the PerformPayment component. Follow the instructions on the previous step, and configure the following variable:



Designing User Components

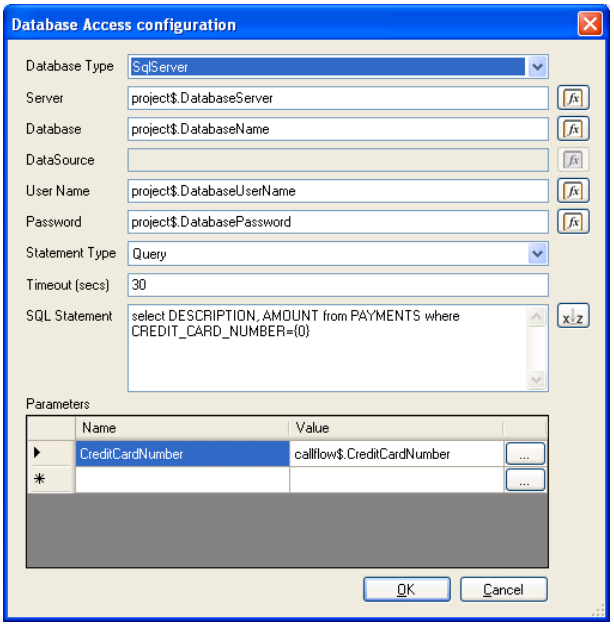
In the following steps we are going to design each user component.

Designing QueryPayments Component

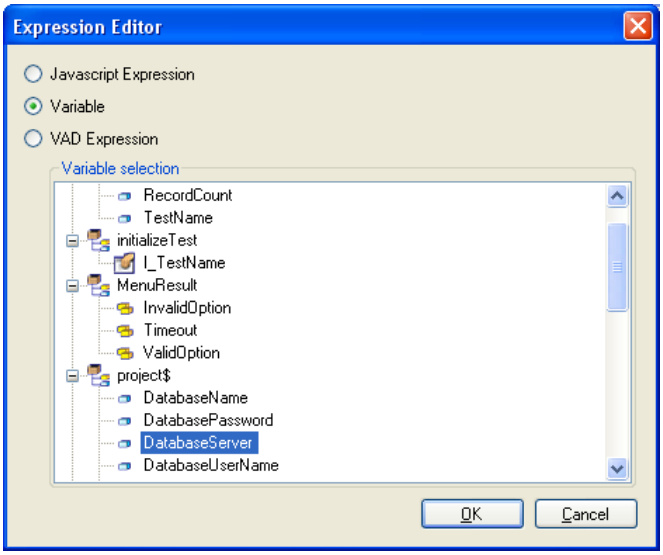
1. From the Project Explorer window double click on the QueryPayments component in order to open it.

2. From the Advanced Components section, drag the Database Access component into the main flow. Select the added component and from the Properties window, click Configure.

3. Input the following information:

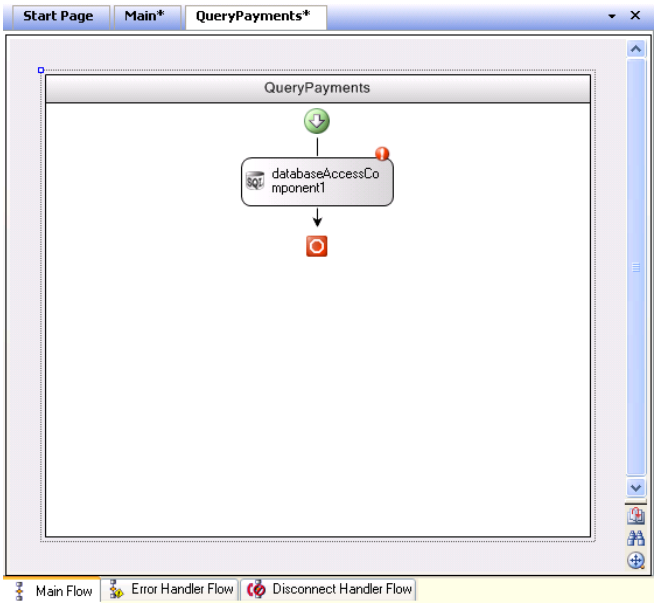


4. To assign a variable to one of the fields of the form, press the Expression Editor Button on the right side of the field, and choose the desired variable. For example, the following figure shows the configuration of the Server field:

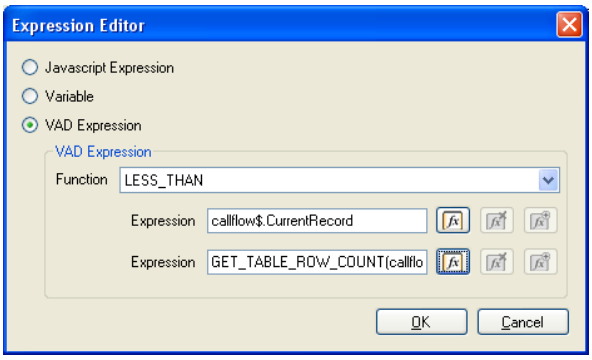


The Project needs to have the variables “DatabaseServer”, “DatabaseName”, “DatabaseUserName” and “DatabasePassword”. They are defined in the Properties Window when the Project is selected in the Project Explorer.

5. In order to assign a value to a parameter, click the button and use the Expression Editor to create the desired expression. The following figure shows the resulting flow at this point:

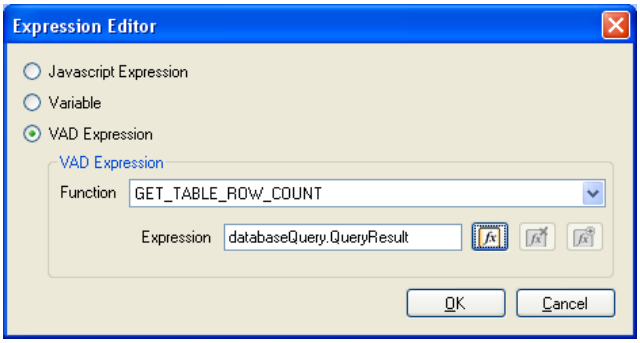


6. We want to loop through the retrieved payments, so we will add a Loop component. From the Control Structures Components section, drag the Loop component and drop it after the databaseQuery component. From the Properties window, change the name of the recently added component to **loopPayments** and the Condition pressing the button. On the Expression Editor, enter the following information:

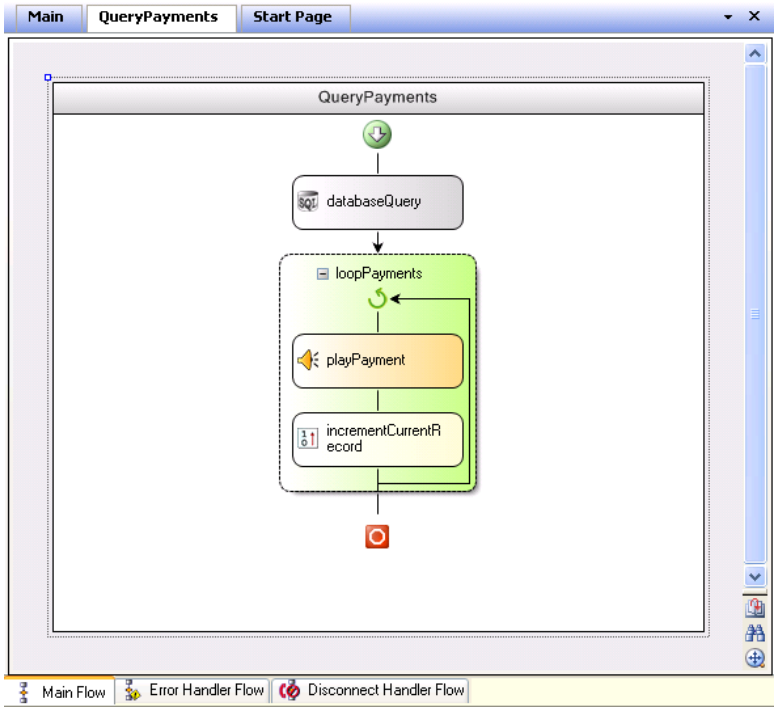


Screenshot 48 - Creating an expression

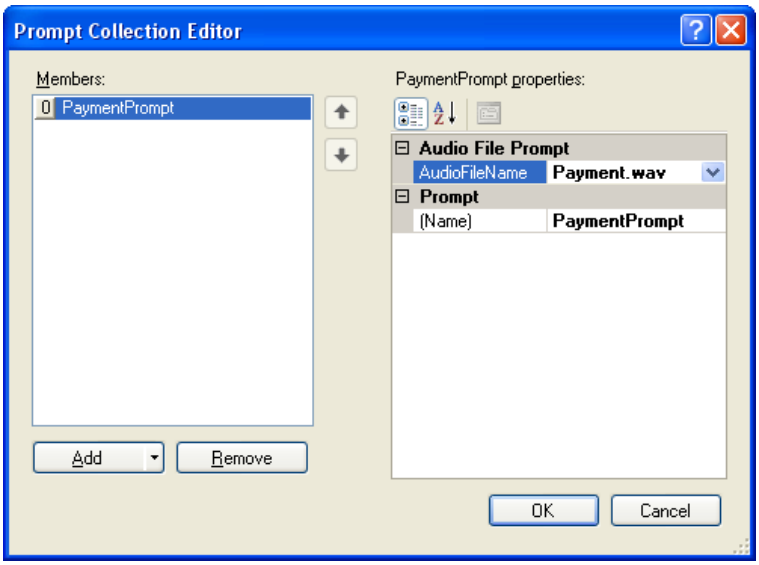
7. To configure the second Expression, press on the Expression Editor button and choose the corresponding values:



The following figure shows the resulting flow at this point:

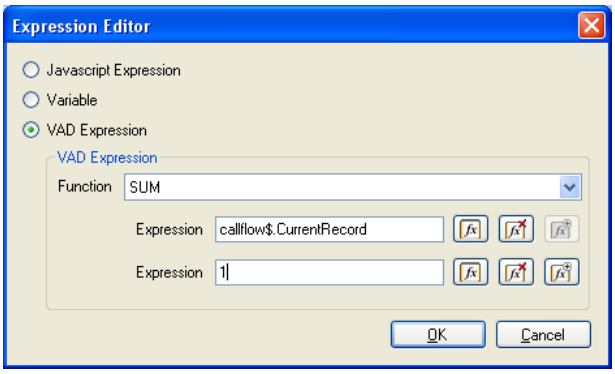


8. Now we are going to configure the Loop component content. We want to play a wav file for each retrieved payment. From the Call Related Components section, drag the Prompt Playback component and drop it into the Loop component. Name it playPayment. Configure the Prompts by pressing the button and entering the following information:



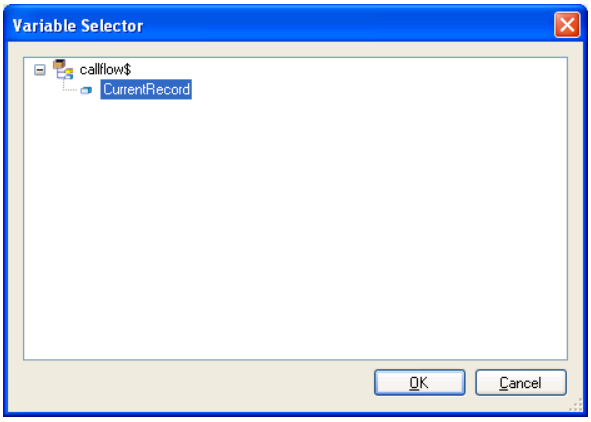
9. Now, from the Control Structures Components section, drag the Variable Assignment component and drop it after the playPayment component. From the Properties window, change the recently added component name to

incrementRecordNumber. On the Expression Editor, enter the following information:



We could use an Increment Variable component to do this, but this is just to show how to create a more complex expression.

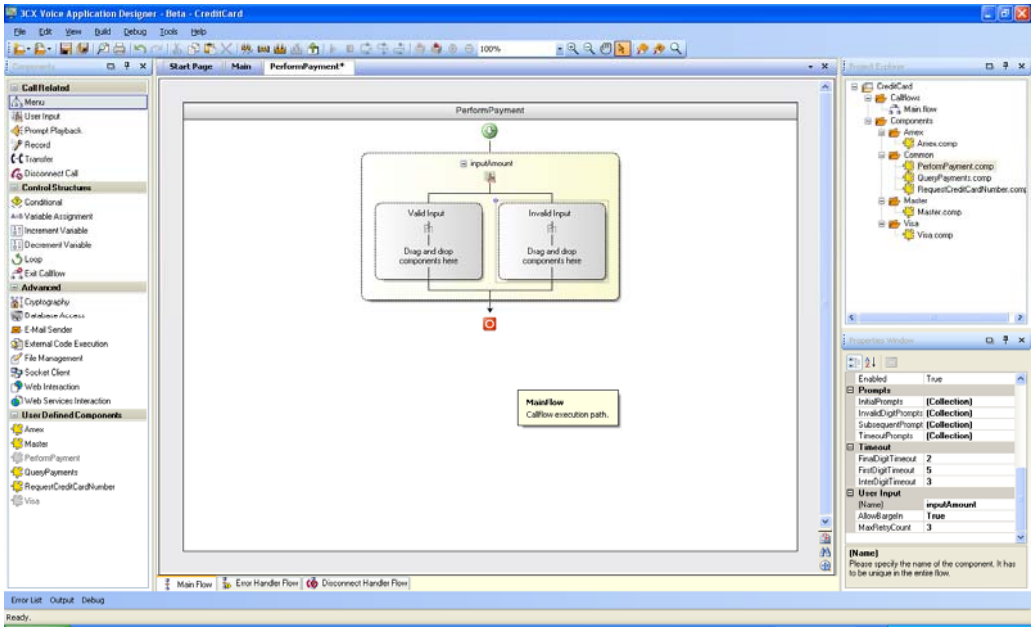
10. Configure the Variable Name by clicking on the button and choosing the following value from the Variable Selector:



We have completed the configuration of the QueryPayments component!

Designing PerformPayment

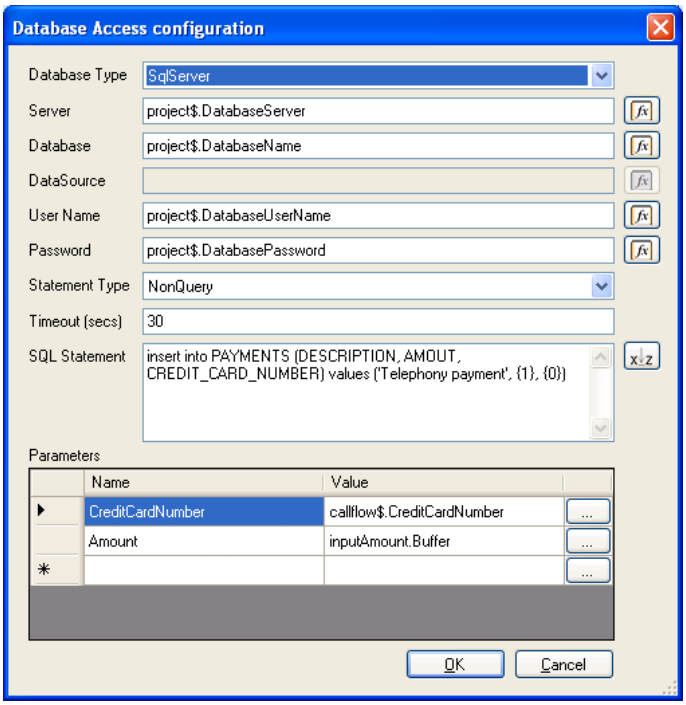
1. From the Project Explorer window double click on the PerformPayment component in order to open it.
2. We want the user to enter the payment amount, so we need to ask for it using a User Input component.
   2.1. From the Call Related Components section, drag the User Input component and drop it into the flow:



3. If the user inputs a valid amount, we want to save the information to the database and play a prompt message to the user.

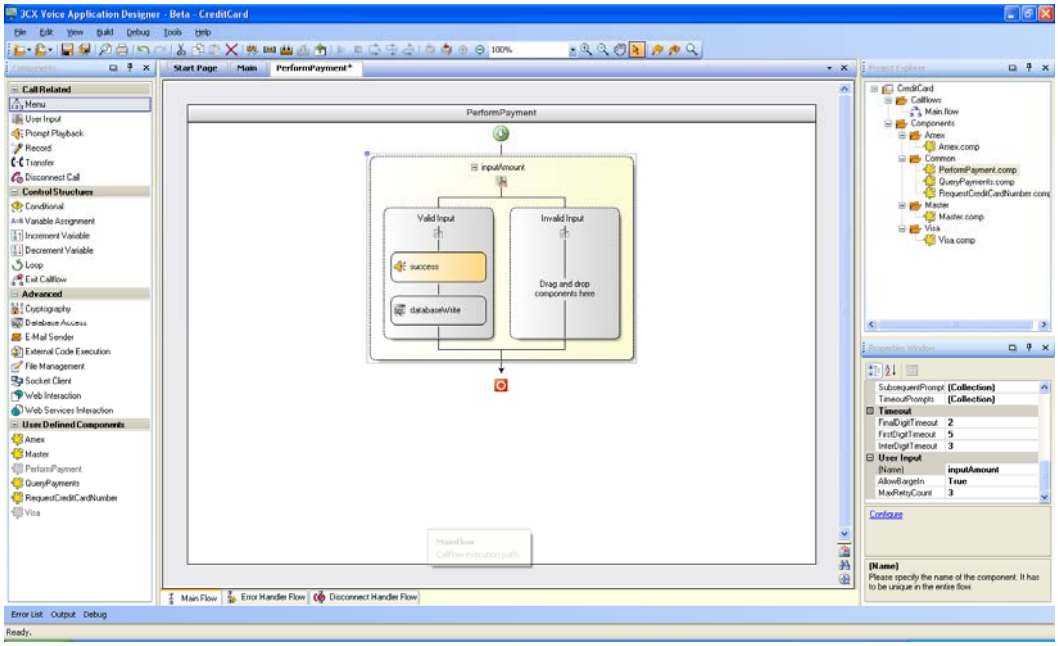
3.1. For the “Valid Input” branch, from the Advanced Components section, drag the Database Access component and drop it into this branch. Select the recently added component, and from the Properties window, press Configure.

3.2. Input the following information:



3.3. Now we are going to add the message to be played if the payment is successfully saved to the database. From the Call Related Components

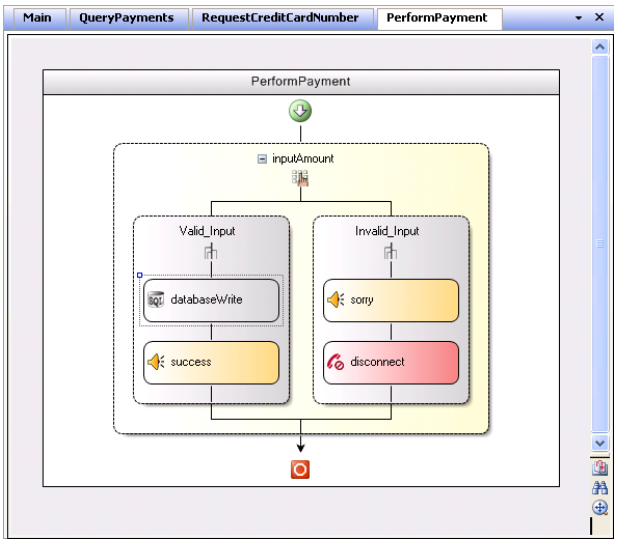
section, drag the Prompt Playback component, drop it into the “Valid Input” branch, and configure the name and the Prompts.



4. If the user does not input a valid amount, we just play a prompt and disconnect the call.

   4.1. From the Call Related Components section, drag the Prompt Playback component and drop it into the “Invalid Input” branch and name it **sorry.**

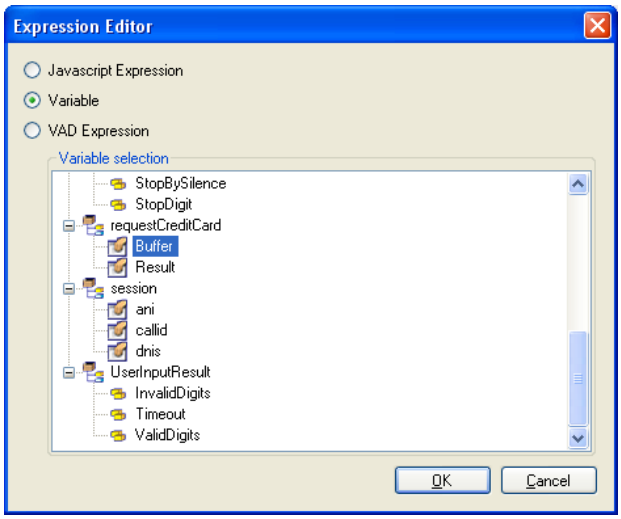
   4.2. From the Call Related Components, drag the Disconnect Call component and drop it into the “Invalid Input” branch. The following figure shows the resulting flow for the PerformPayment component:



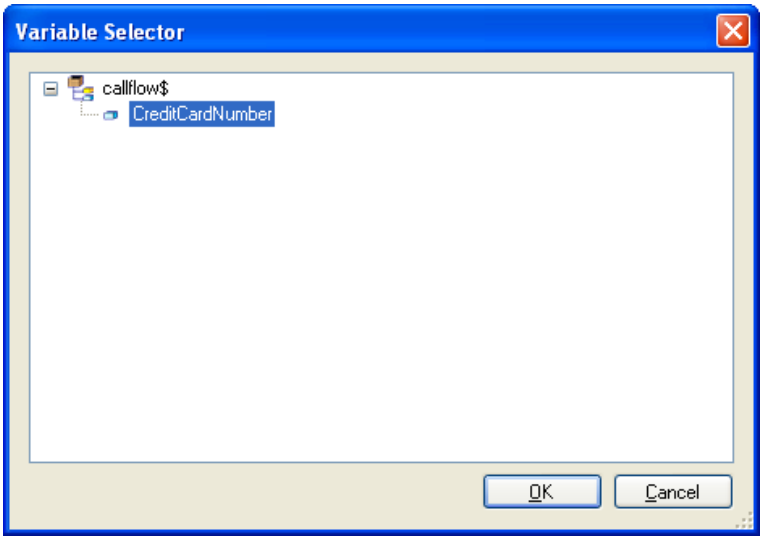
We have configured the PerformPayment component!

Designing RequestCreditCardNumber component

1. From the Project Explorer window double click on the RequestCreditCardNumber component in order to open it.
2. We need to ask the credit card number to the end user, so we need to use a User Input component.
   2.1. From the Call Related Components section, drag the User Input component and drop it into the flow.
3. If the user enters a valid credit card number, we just want to store it to a public output property, so our parent will be able to retrieve it.
   3.1. For the "Valid Input" branch, from the Control Structures Components section, drag the Variable Assignment component and drop it into this branch. From the Properties window, change its name to **storeCreditCardNumber**. Configure the Expression by clicking on the button an setting the following information:



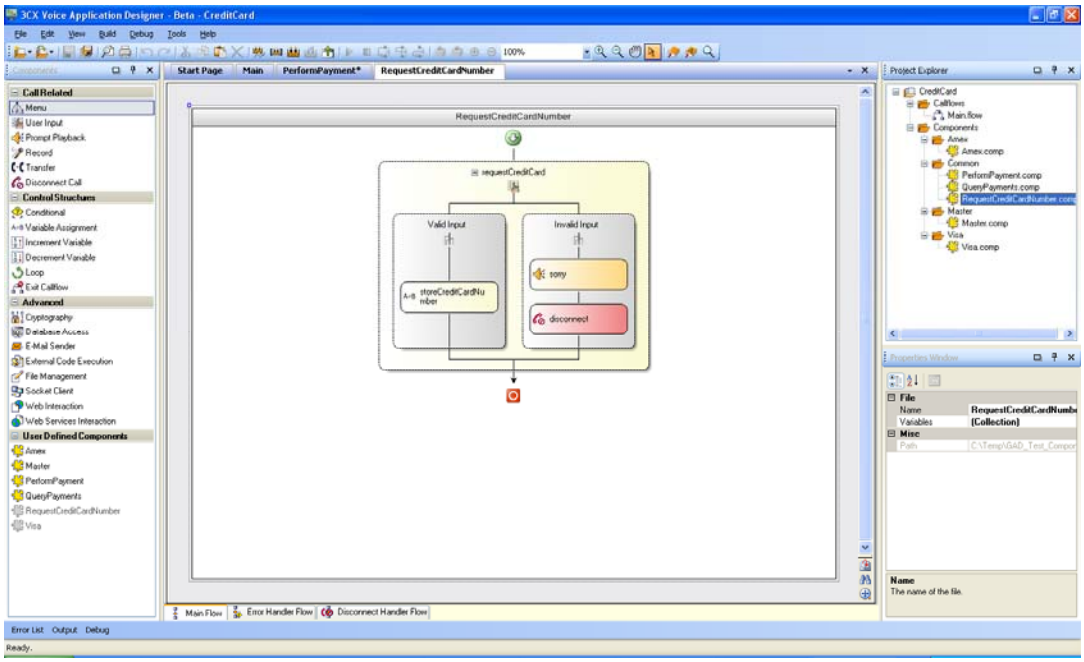
   3.2. For the Variable Name, press the button and choose the following variable:



4. The buffer will be stored into the “CreditCardNumber” public output property.
5. If the user does not enter a valid credit card number, we will play a prompt message and disconnect the call.

   5.1. From the Call Related Components section, drag the Prompt Playback component, drop it into the “Invalid Input” branch and name it **sorry.**

   5.2. From the Call Related Components, drag the Disconnect Call component and drop it into the “Invalid Input” branch. The following figure shows the resulting flow for the RequestCreditCardNumber component:

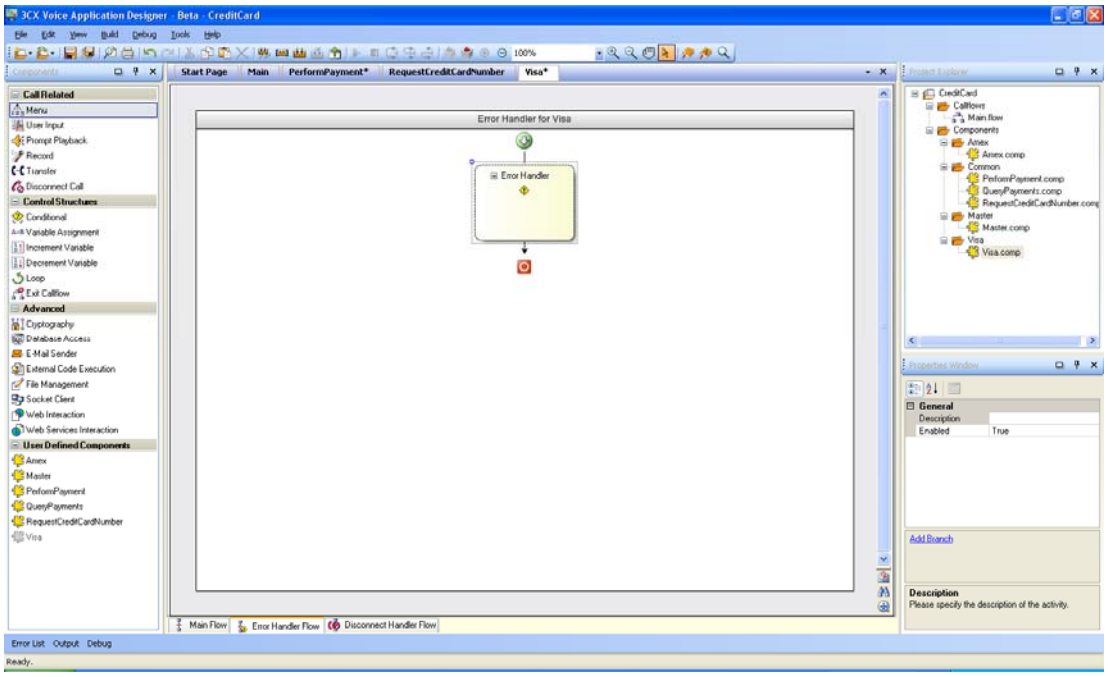


We have configured the RequestCreditCardNumber component.

Error Handler for the Visa Component

As an example, we are going to configure the Error Handler Flow for the Visa component.

1. From the Project Explorer window, open the Visa component by double clicking on it. Then select the Error Handler Flow from the tabs located at the bottom of the designer:

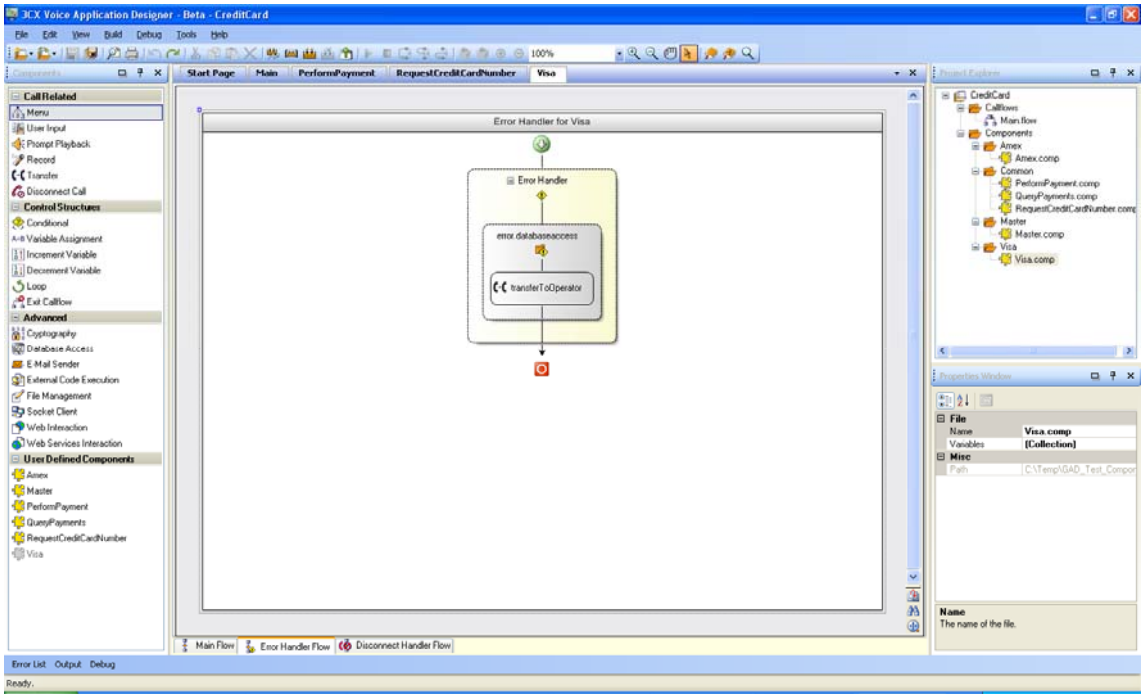


2. We will configure the error handler for database access errors, transferring the call to an operator in the event of such an error.

   2.1. On the Error Handler object, right click and select the “Add Branch” menu command.

3. Now we are going to configure the components for this branch. First from the Properties window modify the **Error Type** for the Error Handler Branch to **error.databaseaccess**. Then from the Components section, drag and drop the Transfer Component into this branch.

4. Now, select the Transfer component and from the Properties window, modify the name to **transferToOperator**. Then change the destination with the proper operator number.



5. If any database access error occurs in the Visa component or any of its children components (PerformPayment or QueryPayments), the call will be transferred to the operator.

We have configured the Error Handler for the Visa component.